University of Southern Queensland

Faculty of Engineering & Surveying

# An Inexpensive Hardware-Based Identification System for Improved Computer Security

A dissertation submitted by

Matthew Quarisa

in fulfilment of the requirements of

**ENG4112 Research Project**

towards the degrees of

**Bachelor of Engineering (Computer Systems)**

**Bachelor of Information Technology (Applied Computer Science)**

Submitted: October, 2005

# Abstract

There is presently a need for better security on computer systems. Usernames and passwords are reasonably secure except for the fact that people cannot be trusted with them. Social Engineers exploit this by manipulating users to extract their credentials. If access to a computer system is protected by a two-factor authentication system, Social Engineers cannot con their way into them.

Using a USB Mass Storage Device, a Hardware-based authentication system was created. This device can be used with *any* PAM-aware application to provide a secure logon system with good portability.

University of Southern Queensland

Faculty of Engineering and Surveying

**ENG4111/2 *Research Project***

## Limitations of Use

The Council of the University of Southern Queensland, its Faculty of Engineering and Surveying, and the staff of the University of Southern Queensland, do not accept any responsibility for the truth, accuracy or completeness of material contained within or associated with this dissertation.

Persons using all or any part of this material do so at their own risk, and not at the risk of the Council of the University of Southern Queensland, its Faculty of Engineering and Surveying or the staff of the University of Southern Queensland.

This dissertation reports an educational exercise and has no purpose or validity beyond this exercise. The sole purpose of the course pair entitled "Research Project" is to contribute to the overall education within the student's chosen degree program. This document, the associated hardware, software, drawings, and other material set out in the associated appendices should not be used for any other purpose: if they are so used, it is entirely at the risk of the user.

**Prof G Baker**

Dean

Faculty of Engineering and Surveying

# Certification of Dissertation

I certify that the ideas, designs and experimental work, results, analyses and conclusions set out in this dissertation are entirely my own effort, except where otherwise indicated and acknowledged.
I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Matthew Quarisa

Q11215969

_____

Signature

_____

Date

# Acknowledgments

I would like to thank my supervisor Dr. John Leis for the direction and opportunities he provided me throughout this year. This would be a very different dissertation without his help.

Also, I would like to thank Jonathon Fowler for his exceptionally good programming skills. He helped me get through many brick walls this year.

Finally I would like to thank my family and friends, particularly my Mum and Dad, for the love and support that has kept me going these past five years.

Matthew Quarisa

*University of Southern Queensland*
*October 2005*

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Background Information

People are not the smartest bunch. No matter how smart security measures for computers become, if they use a password, some user will give it out. It is a concept that many users can simply not grasp; "DO NOT GIVE OUT YOUR PASSWORD TO ANYONE". Now that seems like a simple enough request, however through social engineering, an alarming number of users will give out their password to a third party without realising what they have done.

When a malicious user knows a username/password pair for a system, they generally can do whatever they please with it. Even if the account is a basic user or restricted one, this can be used as a "foot in the door" to launch a further attack on the system from within the system itself.

Furthermore, no matter how good the security of a system is, it is only as good as the password protecting it (*Microsoft TechNet: 10 Immutable Laws of Security* 2005). If users are not forced to use reasonably secure passwords then the security of the system cannot be guaranteed. There also are concerns about password expiration policies of some enterprises; some requiring users to change passwords every month in cases. While seeming to increase security, requiring these password changes so often can result in less

security as users may begin to start jotting passwords down on post-it's or in diaries; "two people can keep a secret, but only if one of them is dead" (*Microsoft TechNet: 10 Immutable Laws of Security* 2005) or once a password is in a place other than the user's head, the password is not secret anymore.

So if a user cannot be trusted to keep their credentials for a system secret, how can the security be maintained? Using a two (or in extreme cases a three) factor authentication system will ensure this is the case.

Most people are familiar with a two-factor system already, namely ATM's. To guarantee that someone is who they are, the machine requires something we know (our PIN) and something we have (our card). There are many other security systems, such as biometric devices that authenticate via fingerprints, our eyes and even our voice. Smartcard authentication is very popular amongst enterprises, especially due to the crypto functions being performed on the card itself rather than in software on the computer.

It is only recently where we have seen the rapid growth of both two-factor or token based authentication and biometric identification. For under $100 it is now possible to purchase a stand-alone Microsoft fingerprint scanner which simply connects to a USB port and allows users to logon to their computer simply by their finger. Some USB storage devices now incorporate a fingerprint scanner into their devices to ensure that if the user either loses or has their device stolen, the data onboard cannot be (easily) compromised. Smart-card readers are also widely available now, allowing people to login to their computers using either a smartcard or a smartcard/PIN pair and thus creating a two-factor authentication system.

There are some pitfalls with these devices. The main drawback is that smartcard based systems require the owner/operator to purchase a smartcard for every user that wishes to use the system. While the cost of a card is relatively inexpensive, it still requires purchasing something that the user probably does not have already.

Biometrics may seem to alleviate this problem; after all, practically everyone has fingers and it's not likely that they will lose them anytime soon. As mentioned by Kay (2005),

the two big problems with fingerprint authentication are that the fingerprint readers can be fooled the majority of the time with gelatin moulds of fingers. Secondly, if someone manages to reproduce a fingerprint either electronically or by creating a mould, there is no way to revoke that fingerprint as it cannot be (essentially) changed (Kay 2005).

## 1.2 Project Objectives

The aim of this project is to implement an authentication system for computers using off the shelf hardware, such as a USB Flash Disk.

It is also expected that investigations and discussions into other security devices available would be made. This would also be extended to security measures and protocols that can be used today.

The project hopes to:

- Evaluate all the popular and influential forms of authentication means for computers at home and in business currently available.
- Research studies done in the past regarding social engineering and password cracking.
- Show the effects on social engineering when a two or three factor authentication system is used instead of traditional username/password pairs.
- Evaluate common copy protection techniques to limit access to a (storage based) security device by a malicious user.
- Create a token based, two factor authentication system using a USB Mass Storage Device, such as a USB Thumb Drive and integrate this into a login manager using PAM modules or a modified GDM.
- Show whether or not the device is a secure, robust authentication device for a computer.
- Discuss implementation of the device on other Operating Systems.

- Describe a network authentication architecture that can take advantage of the hardware device to help increase overall security.

## 1.3 Overview of the Dissertation

This dissertation is organized as follows:

**Chapter 2** details what Social Engineering is and how it affects computer security. Approaches hackers use to gain information and strategies that can reduce the effect of Social Engineering are explored.

**Chapter 3** describes the current authentication systems that are available presently for both home and enterprise computer systems. This chapter pays particular attention to the field of Biometric Authentication including systems such as Voice and Facial Recognition.

**Chapter 4** discusses processes and technologies that are used in mainstream digital media to prevent duplication. Methods of circumventing some of these protections are also presented.

**Chapter 5** shows how the Pluggable Authentication Module (PAM) system can be used to allow an application to support multiple authentication subsystems, even if not explicitly supported by the original application developer.

**Chapter 6** discusses the different approaches that were available in designing the security device.

**Chapter 7** describes the actual development of the USB security device, outlining each component of the system.

**Chapter 8** discusses the extension of the security device, adding features that would appeal to enterprise users.

**Chapter 9** concludes the dissertation and the overall results of the project. It details the project goals that were achieved and discusses ones that were not. Finally it discusses further work that may be commissioned for the device at a later date.

# Chapter 2

# Social Engineering Study

This chapter discusses what Social Engineering is and how it affects security. Techniques that hackers use to gather information and exploit people are also mentioned. Finally, strategies that can help reduce the effects of Social Engineering in a business are presented including the use of multi-factor authentication systems, similar to the device created in this project.

## 2.1 Background

Social Engineering (SE) is the practice of obtaining confidential information by the manipulation of legitimate users (*Social engineering (computer security) - Wikipedia, the free encyclopedia* 2005). SE has been made possible due to username/passwords and the manipulation of users who are supposed to be protecting that data. While business's implement password security policies, it is mostly the users who are to blame for networks being compromised. This is especially true for systems which have strong security but users pick weak passwords (*Microsoft TechNet: 10 Immutable Laws of Security* 2005). It goes onto say that many people simply use no password as their password.

There are three forms of SE; they are, in order of occurrence:

- Posing as IT staff or high-level management and telephoning users.
- Posing as high-level management and telephoning IT staff.
- Develop a relationship with a user or IT staff member and extract the information directly.

The first, being the most common, works by the hacker telephoning a person at a business and pretending to be from IT or someone higher up in the business. They may then pretend there is a server issue and they are restoring the backups, but they need the users' name and password to verify that their files have been restored correctly. The user will unknowingly give up these credentials thinking they are ensuring their own data is not lost. Meanwhile, the hacker is using these credentials for whatever purpose they wish.

The next form is where a hacker will pose as high-level management, calling the IT staff directly and pretending to need some credentials due to some urgent matter. For example, a hacker might telephone the IT staff and pretend to be a manager in a panic because he cannot get access to his email and download his PowerPoint slides for a presentation he is about to give. The hacker would then possibly threaten the IT workers job if he does not reset the manager's password at once.

Finally, the third form of SE, and by far the most deceitful form of social engineering. This is where the hacker will develop a personal relationship with a user or IT staff member, with the sole intention of "sweet-taking" the information out of the user directly. This is deceitful because the user may develop a serious relationship with the hacker and when the bond is broken (usually because the hacker has gotten the information they need and has then disappeared), the user or in this case, the victim, may become depressed or may even go as far as closing themselves off from social activity.

A good social engineer has a few important qualities, the three most important ones are:

- Good at acting/improvisation.

- Good at reading people.
- Good researching abilities or information gathering techniques

If a social engineer is to be successful more times than not, they must be good at acting or improvisation. They must be able to think on their feet in high pressure situations. This becomes ever more important in a situation such as gathering information over the telephone. A hacker may be trying to extract someone's credentials from a helpdesk operator. This operator may become suspect to the identity of the person requesting the credentials. The hacker should not just hang up the phone and leave as this would remove the doubt in the mind of the operator of the legitimacy of the request. It is by far much better if the hacker can make up a plausible excuse and leave the conversation much more calmly. One exit strategy can be simply the hacker pretends their credentials are now working and thus there is now no need to have the password reset.

The ability to read people is another important quality for a social engineer. They must be able to look at a mark or victim and know almost immediately what approach will work the best to extract the information they need. One example of this is if the hacker is posing as a courier while gathering information about the business, they may notice the secretary is having trouble with their computer. This may show that the secretary has poor computer skills and probably would not question any direction given by someone posing as IT staff. So just by reading, who is possibly, the first person they see upon entering a building, the hacker has a good chance of obtaining a username and password which can be used to gain entry to the computer system and possibly launch further attacks from the inside.

A social engineer is only as good as the information they know about their mark. It is very important that they have a basic understanding of how the organisation works internally. To improve their knowledge of the organisation, a hacker may phone the business, poking for information related to something they need. This is usually met with limited success as employees may start to suspect something is not quite right when lots of questions are being asked. Another way is simply to use the internet to find all the information available. This is once again usually of limited use as all the information on corporate websites is only for the public's interest.

Failing to obtain the information using one of the above techniques, a social engineer will resort to "dumpster diving". This is where they will simply go through an organisations waste bins looking for anything that can help them sound more credible while they are conning. This works more times than not because of the lax security that usually surrounds enterprises' waste bins. The social engineer will look for anything that can help, this includes:

- Draft reports
- Memos
- Letters
- Meeting minutes
- Internal phone directories

Internal phone directories are highly sought after, especially in large organisations. Most of the time, these will have the persons full name, position in the company, and phone number. This allows the hacker to target their con directly to the person to whom it is suited, rather than going through layer and layer of helpdesk operators and assistants, possibly making a mistake through one of the levels as the call is escalated.

Managers seem to think that if their staff are put through an IT introduction and they touch on the importance of keeping usernames and passwords secure, then there is no need to worry about computer security. They are so wrong. A survey conducted in the UK back in 2003 showed that "Ninety per cent of office workers at London's Waterloo Station gave away their computer password for a cheap pen, compared with 65 per cent last year" (Leyden 2003). That is a staggering statistic. The same survey went on to say that workers were asked bluntly "What is you password?" (Leyden 2003) and 75 percent of the employees gave it out immediately.

In the year following, another unrelated survey was conducted using a similar technique. The results this time were just as alarming; "70 per cent of people would willingly trade their computer password for a bar of chocolate." It then went on to say that "... 34 per cent would give away their password for nothing!" (Grammeman 2004).

## 2.2 Security Considerations

It is very important that management take educating their employees about social engineering seriously. The staff are the ones who will have an interaction with the hacker most of the time so its critical that they can identify when they are being conned and take appropriate action.

There are two sides to social engineering assaults, physical and psychological. Physical attacks are catered for in most cases for many businesses, but they fail to raise awareness to the psychological ones. Granger (2001) talks about removing the responsibilities of staff to make judgment calls in certain situations. That way, if the hacker asks for something that is against the security policy, the employee must deny the request. It should also be part of employee training that they are taught to identify SE attacks. This is by far one of the most effective techniques against SE as every employee is actively watching out for any suspicious activities.

Physical security is not that hard to maintain effectively. Obvious things such as ID cards for every employee and if security is paramount, ensuring that these are checked and verified upon entering and exiting the building (Granger 2001). Ensuring that all doors that require security cards to access are kept closed at all times. This may require employees once passed through the doors to wait a moment while the door shuts to ensure it locks.

Network points should also be secured. This is usually done in large businesses by disabling the port on the switch or router directly. Smaller businesses may simply disconnect the port from the patch panel in their equipment racks. If this is not done a hacker may simply be able to walk in and plug a laptop into the network and gain access. Wireless networks are another concern for network security. They pose a much larger problem for the business as there is no medium that can be locked down like a wired network. This implies that strong encryption must be used over wireless links, not using Wired Equivalent Privacy or WEP; it is by far not secure. Wifi Protected Access (WPA) can be considered secure when bundled with a central RADIUS authentication server.

Still the transmissions can be intercepted and if data is highly sensitive then wireless networks should be banned. This can be enforced by deploying access-points with rouge network scanning features. These devices can then jam or disable the unauthorised network automatically (*Cisco Self-Defending Network - Cisco Systems* 2005).

Physiological attack prevention is more difficult to implement effectively, but by no means impossible. This can only be achieved by training employees on a regular basis. Most companies have a security talk or presentation for new employees but leave it at that. They do not have ongoing, updated sessions and they definitely do not keep social engineering in the minds of their employees. If staff become lax about security, even for a few days, this is more than enough time for a social engineer to compromise many areas of a company.

The other important concern raised is that "A computer is only as secure as the administrator is trustworthy". Anyone who has administrator privileges on a system can do whatever they want and cover their tracks so no one can detect them. There is no security on a network if the administrator cannot be trusted. This is very important, especially for larger organisations which may have huge amounts of confidential data residing on servers. In these cases, background checks are usually performed on people who have access to critical data. Of course, not every business has the resources to check into every new administrator but if security is paramount, it should be carried out. If a business uses some form of two-factor authentication for its users, rouge administrators will have a harder time trying to spoof accounts, especially if the business uses biometrics. It will not (usually) help keep files anymore secure as administrators in most situations have full access to all files on any given server.

Another related Microsoft TechNet article "10 Immutable Laws of Security Administration" talks about security concerns and observations that are more targeted towards administrators than users. One such is "Law 2: Security only works if the secure way also happens to be the easy way" (*Microsoft TechNet; Security Administration: 10 Immutable Laws* 2005). There is no point trying to convince management to approve new authentication systems if they are too expensive or would require too much training for the users to interface with effectively. Most of the time, businesses will regard the secure way to be username and passwords. These are still the dominant form of authentica-

tion because in essence they are the easy way; especially when you consider that nearly every operating system ever comes straight out of the box with everything necessary to authenticate based on passwords. The other law that is of interest in the context of this project is "Law 6: There really is someone out there trying to guess your passwords" (*Microsoft TechNet; Security Administration: 10 Immutable Laws* 2005). No matter how secure your network, nearly all are susceptible to password-guessing attacks. While good passwords that are changed regularly can afford good protection from this type of attack, it is not 100% secure. If a business uses a two-factor based system, such as a smart card, or a form of secure biometrics, they will be effectively immune to these attacks.

## 2.3 Chapter Summary

This chapter clearly showed how Social Engineering can undermine the security of any password protected system. The three most common forms of SE were discussed along with the qualities that make a good Social Engineer. These qualities enable them to extract priviledged information out of users over 70 per cent of the time.

Methods of reducing SE were also discussed. There are two aspects to social engineering attacks; these being physical and psychological attacks. Most businesses have little or no trouble implementing protocols for reducing physical attacks, yet neglect the psychological ones. Sometimes these psychological attacks can be reduced by simply implementing policies where by users can not make judgment calls on certain issues; if the policy denies a certain type of request, then the user must adhere to that.

# Chapter 3

# Current Authentication Methods

This chapter details various authentication devices used presently on computer systems. Smart Cards are briefly discussed before a detailed look into Biometric Authentication is undertaken. This includes evaluation of a number of Biometric Authentication Systems such as Eye Scanning and Facial Recognition.

## 3.1 Two and Three Factor Authentication

When someone says do you have a two factor authentication system in place or have you ever used a two factor authentication system, most people will shrug their shoulders and say "I don't know"; yet almost everyone has at some stage in their life used a two or even three factor authentication system without even knowing it.

Debit or credit cards are by far the most common form of two factor authentication. To withdraw money or to pay for purchases you need two things; something you have, this being the actual card; and something you know, this is of course either the PIN or the cardholders signature for debit and credit cards respectively.

The same principle applies to cheques. For a transaction to be successful, two things are required, a cheque and a signature. Cheques however, can become three factor authentication based when a bank or a cashier asks for some identification, such as a

drivers license. This is done to further ensure that the person writing the cheque is indeed who they claim to be.

In computer security terms, the base principles of two-factor authentication are still the same; something we have and something we know. This can be extended to three-factor authentication by the addition of some biometric identification, such as a fingerprint. Now to access a system you need something you know, something you have and something you are.

## 3.2 Smart Cards

Authentication-wise, these work in a similar fashion to debit cards; you need the smart card and usually a PIN to gain access to a system. Smart cards go one step further than traditional magnetic strip cards by having a small embedded processor integrated into the card. These usually perform cryptography functions using a small amount of on-board memory. The point of this is to protect the actual key or key code on the card from whatever is requesting authentication.

This abstraction makes smart card authentication quite secure. Since there is no way to access the key on the card (short of scraping the top off one of the RAM chips and using an electron microscope to read the data atom by atom!), the key cannot be compromised. If something requests authentication, it can send a challenge to the on-board processor and the processor can check the key before returning a success or failure to the initial challenge.

## 3.3 Biometrics

While smart card authentication is quite good in itself, it is coming under threat by another form of authentication, this being biometrics. *Social engineering (computer security) - Wikipedia, the free encyclopedia* (2005) defines biometrics as “the study of automated methods for uniquely recognizing humans based upon one or more intrinsic physical or behavioral traits”.

Within the vast field of biometrics there are a few predominant authentication systems:

- Fingerprints
- Hand geometry
- Voice recognition
- Eye scanning
  - Iris
  - Retina
- Facial recognition

### 3.3.1 Fingerprints

Identifying a person by fingerprints has been a trusted means for many years. Police by far are the biggest users of fingerprint identification, using them almost exclusively (until the advent of DNA identification) for placing criminals at crime scenes when no witnesses were around. Large businesses have been using fingerprint based authentication systems for years now with good results. Now though, this technology is making its way into end user's equipment. For example, Microsoft market a line of fingerprint authentication peripherals such as keyboards and mice with fingerprint readers built right into the device. Many PDA's and laptops are now available with fingerprint scanners integrated directly into the device.

While this sounds great for consumers who wish to secure their systems better, there are some drawbacks to these "consumer-friendly" scanners. Firstly some are slow; this is also dependent on the computer's specifications and secondly they are more susceptible to counterfeit fingerprints.

A Japanese firm (find out who again) decided to test some consumer-grade fingerprint readers. They began by lifting a fingerprint off one of the reader's scanning plates. This was not done by dusting for the print and using some contact adhesive sheet, but by taking a photograph of it! They then used the photograph of the print to create a

gelatinous mould of the finger. Once the finger was created, they set out to test the fingerprint readers. They were able to fool the readers and gain entry to the computer system 80 percent of the time. This shows that a basic fingerprint scanner is by no means foolproof, especially if it protects something that someone wants to get their hands on badly.

Not wanting to have their system's fooled or opened under wrong circumstances, the Department of Energy in America has pulse or heart beat detectors along with pressure sensors in their finger and hand authentication systems. This is to combat the use of finger or hand moulds or to prevent a criminal from cutting off someone's fingers or hands to gain entry to the facility. If someone does try to use a severed limb, the system will fail to detect a pulse and also the limb may seem flatter due to the loss of blood and as such, will fail to authenticate the user.

### 3.3.2 Hand Geometry

Hand geometry or hand authentication has been used for over 20 years. It is a reliable, non-invasive form of accepted authentication. How it works is by measuring the distance between known points on the palm of ones hand. These are quite distinctive from person to person and as such form the basic of hand geometry authentication. Many public places use this form of identification, airports being one large users. Hand-prints help speed up business people upon re-entry to the country. They simply swipe a card and scan their hand and they are admitted without the need for an official to inspect credentials.

Another use for this technology is in more harsh work environments, such as mechanical workshops or mines. Since finer details such as fingerprints are not needed, employees with dirty hands can still be authenticated successfully. Probably the biggest user of hand identification is the US Prison System. Not only are the staff scanned for access, but also the inmates and visitors. This means that at any one time an audit can be conducted on the facility and everyone accounted with a relatively high accuracy. This is especially important in situations such as a riot by inmates or an evacuation of the facility.

### 3.3.3 Voice Recognition

Voice recognition technology is an ever changing field in biometrics. The two main forms are authentication and dictation or text recognition. It is quite difficult for a computer system to understand a user when they are talking and translate the spoken words into text, especially with the correct punctuation and emphasis. This is due to the many subtleties in human speech and as a result makes naturally speaking to a computer a very mean feat to accomplish.

Recognising a small phrase or group of words accurately is vastly easier to do. By working on a much, much smaller subset of words, say four or five, algorithms can distinguish with a good degree of certainty any given person. Many large businesses use this technology to protect access to computer rooms and vaults. Once again, because this is a non-invasive authentication method, end users generally have no reservations about embracing it.

### 3.3.4 Eye Authentication

Eye authentication is a relatively new system in mainstream business. It has been around for many years but its use was severely limited to large businesses that have something quite secure to protect or limit access too. Many people have concerns about using an eye scanner. While some systems allow users to stand a few feet from a scanner or camera, other more secure systems usually require users to place their faces in a guide or position their eyes directly in front of a scanner like a pair of binoculars. This is an invasive check; some people fear this and as a result, seem to shy away when given the option to use it.

There are two methods of eye authentication, the first being retina based scanning. This involves scanning the retina; using the blood vessels as unique identification markings. Due to the need to scan the back of the eye, a user needs to be standing quite close to the scanner and must keep the eye as still as possible. The results are quite accurate though and as such have been used for over 15 years with only minor issues; namely that retina patterns can change over time. This may be due to blunt trauma to the eye

or the head or simply just aging. As a result of this it is important that the system is kept updated with ongoing "refresher" scans.

The second method, iris based scanning, is a better form of eye authentication. In this method, the iris is photographed and the structure analysed using computer algorithms. There are two advantages to this system. Firstly, it is less invasive than retina scanning. All that is needed is a good quality camera and the iris pictures can be taken from a short distance from the user. This also means that iris scanning can be done sometimes without users knowing, good for tracking people through a public place. Secondly, a person's iris does not change over time, similar to a fingerprint, and as a result does not need to be retrained on a regular basis. Furthermore, identical twins can be identified accurately using iris analysis, making unique and accurate authentication possible when fingerprints and sometimes retina scanning cannot.

### 3.3.5 Facial Recognition

Finally we come to facial recognition. This is a relatively new field when compared with much older technologies such as fingerprinting and retina scanning. Facial recognition uses pictures of a persons face as the basis for a uniqueness algorithm. An algorithm attempts to map points on a face which do not change with either age or with disguises. This means good facial recognition systems should be able to identify someone with glasses on or who has changed their complexion to fool security.

The big problem with these systems is they require a lot of computing power to check faces, especially when there are many cameras all feeding many frames per second. To make matters worse, each frame may have several people in view, further complicating the system. Performance issue like this can be handled in one of three ways usually:

- Reduce the video feeds frame rate or pause it entirely
- Reduce the image quality
- Increase the computing power

Reducing the frame rate is usually the preferred choice. Image quality is not sacrificed and as long as there is no need to capture all the extra intermediate frames that we are now ignoring, we can save on storage space when archiving the footage.

Reducing the image quality is usually a bad idea. Doing this usually results in a reduction in recognition accuracy. Unfortunately there are some situations where this has to be done, such as when a camera is mounted in a remote location and the only communications infrastructure available is either a dialup or low-speed wireless modem. In a larger, more mature area, such as a stadium or airport, reducing the image quality will ease transmission and storage of the footage; it also will ease the load on the facial recognition system as less data needs to be processed in any given video frame but the accuracy will suffer. The most common way to reduce image quality is to compress the image, usually by use of a video compressor/decompressor (codec). Most video codecs are lossy, meaning that once the camera compresses the video for transmission, elements of the video are lost and cannot be recovered.

Increasing the computing power to handle the load is the other solution. This is usually only done if it is of the utmost importance that faces be matched accurately and timely. An Olympic Game is the sort of magnitude that would warrant a large server upgrade; traffic and street cameras in a town are something that would not usually be upgraded.

There is some hope though. A new breed of "smart" cameras are starting to appear. One company in America, Oxford Microdevices, has cameras that are outfitted with "inexpensive but very powerful Digital Signal Processor chips" (Cringely 2005). These cameras then can actively identify people from many distinguishing features. This means no more monitoring hundreds of video screens and no more server farms to do analysis from the cameras. When a camera finds something of value it then alerts a command centre and begins to stream the video. That's the brilliance of this system, when you deploy 100 cameras, you are also deploying the processing power needed to monitor those cameras. Do not think that because the camera has some intelligence and it uses a few cheap DSP chips it is more a gimmick than a feature; most of these are running at "more than 50 gigaflops - 50 billion floating-point operations per second" (Cringely 2005).

## 3.4 Commercial Authentication Systems

There are many different authentication means available for computer systems, but only a handful are available for mainstream markets. Furthermore, only a small number of these work under operating systems other than Windows. Smartcards seem to be the only universal operating system authentication device; working in most OS's and many embedded devices.

### 3.4.1 Cryptocard

Cryptocard have bridged the operating system gaps with their UB-1 USB Authentication tokens and their Linux-supported smart card software. The USB token is really a smart card and a smart card reader integrated into something the size of a USB thumb drive; however these devices are not cheap, running in the order of $300 US for a five pack. That is not the final cost though, as five licences on the authentication software are required. When purchased in a starter pack, the five USB tokens and the software with licences cost $500 US. Additional licences cost in the order of $1250 US for 25 licences; not overly expensive for a large business, but definitely for a smaller one (*CRYPTOCard Corp* 2005).

### 3.4.2 Dekart Logon

Dekart is another company who specialise in authentication solutions for home and business users. Their authentication system is called Logon and can use basically any media or authentication device to secure a system. It has no Linux support and licences are about $40 US for personal or business use. The Dekart system may be less secure than the Cryptocard system, especially when using the USB flash drive or CD/floppy disk authentication methods; as the tokens may be copied easier off these devices than the dedicated smartcard-like architecture of the Cryptocard system. However, for Windows-based authentication, the Dekart system supports an impressive number of devices including hardware USB tokens, Smartcards, and any other generic storage devices such as USB thumb drives, CD's etc. It can also interfaces with a number of

Biometric verification devices (*Free download - secure login for Windows - USB flash disk/smart card, biometric login with Dekart Logon* 2005).

## 3.5 Chapter Summary

This chapter discussed the main types of two- and three-factor authentication systems currently available. Smart cards are a reliable form of authentication and have the added advantage of being non-invasive for the user.

A few common forms of Biometric Authentication were then discussed. Fingerprint scanning is the latest form with devices now cropping up, sporting fingerprint authentication. The problem with these devices though is they can sometimes be tricked by false fingerprints. Hand geometry is a valid form of authentication but lends itself to larger security installations such as vaults, rather than portable devices such as PDA's.

User identification by voice was also mentioned. Used mainly for phone-based services, such as telephone banking, it is another non-invasive form of authentication. Conversely, one of the Eye scanning techniques is invasive; this being retina-based scanning. This method requires users to stand quite close to the scanner and remain as still as possible. The other Eye scanning technique uses the iris to distinguish between users. This method is much less invasive that retina-based scanning which is better for the user. It is also able to discriminate between users much better than other biometric methods, so well in fact that it can distinguish between identical twins. Finally Facial Recognition was discussed including technological advances in processing ability.

# Chapter 4

# Copy Protection Techniques

This chapter discusses copy protection techniques used on digital media. A few common copy protection systems are detailed. Methods of defeating these protections are also given. In addition, the application of copy protection to the USB security device is mentioned.

## 4.1 Duplication Prevention

There are many different copy protection systems in place today to protect digital media. Music compact discs and computer programs or games are the biggest users of copy protection. Companies do not want people duplicating their media; copying songs, movies or games and reducing their profits.

The two big names in CD/DVD copy protection are SafeDisc and SecuROM, owned by Macrovision and Sony respectively. These two methods, along with many others, are used to secure primarily PC software, especially game titles, from unauthorised duplication.

### 4.1.1 SafeDisc

SafeDisc protection is a software based system whereby a digital signature is embedded into the disc by a laser beam. Then the executable for the software is encrypted with a loader program. When executed, the loader checks for the authentication signature, and if it is available, decrypts the executable and runs it.

The actual program executable is stored in a .icd file for SafeDisc 1. The main executable is merely only an authentication wrapper that does the signature checks and the decryption of the .icd file. Under SafeDisc 2, the encrypted binary .icd file and other key support libraries are embedded in the loader or authentication wrapper. The wrapper or loader itself is encrypted as well, decrypting itself upon execution. Once this has occurred it then loads a kernel driver named secdrv.sys. On a Windows NT-based operating system, this driver is used to detect the presence of a debugger. If one is found or it detects the presence of software breakpoints in kernel functions, the decryption of the loader will fail and prevent the decryption and execution of the protected application. When the loader has been decrypted successfully, it checks the CD/DVD for the digital signature "using direct SCSI operations on the CD driver" (*SafeDisc FAQs* 2002). Once the key is verified, the .icd is decrypted and execution is transferred over to it.

This sounds impressive but it is really just an inconvenience for someone who wishes to duplicate a game or a program. Anyone with a CD/DVD recorder that can burn in disc-at-once (DAO) RAW mode and software that supports this (such as CloneCD) can defeat SafeDisc protection. Furthermore, people have gone onto creating software that can remove the SafeDisc protection from programs.

### 4.1.2 SecuROM

SecuROM uses a similar protection technique to SafeDisc, however the software that is being protected is usually aware of this, unlike some SafeDisc applications. New versions of SecuROM also support "Trigger Functions", similar in function to the features found in the SafeDisc API . These allow programmers to place many authentication requests,

all of which can be customised in any way, throughout their program. This attempts to create a more secure copy protection system, as opposed to single check systems (*CD Media World - CD/DVD Protections - CD/DVD Copy Protections - CD/DVD* 2005). Most of these systems also have facilities in place to check whether the media currently in use is a mastered disc or recordable medium, such as a CD-R.

While SafeDisc protection embeds a digital signature into "defect sectors" which are designed to be hard to read yet can be read by a good CD recorder, SecuROM 4.8 and greater use a different approach. Data is stored on an optical disc in a large spiral formation, starting off small from the centre of the disc and ending up large when near the outer edge. Since the spiral varies in size throughout the disc, there are measurable differences in read times for the CD drive to read a sector from one area to another. SecuROM exploits this fact by using the time taken to read two sectors as the key. If the delays do not match the set pattern, the disc is deemed copied or corrupt.

How this protection can work is because CD-R's have this spiral track already embedded into them when manufactured. The optical dye is impregnated into this track and changes when the CD is recorded and thus, the time taken to read one sector to another is fixed. The original track cannot be duplicated because the target medium already has an invalid track before the CD is even written. It is impossible to make exact 1-to-1 copies of SecuROM discs.

### 4.1.3 Laserlock

Another company, MLS Laserlock International, has similar based CD/DVD protection systems however they use distributed corrupted blocks on the disk, 20MB of them on a typical CD-R. In these corrupted blocks, they hide what they call "good data", presumably the physical signature for the encryption software. They also mention support for protecting memory cards, a technique which may be useful on a USB storage device. Unfortunately, this seems to be a relatively new product for them and as such their site says nothing more than "Available Soon".

## 4.2 Duplication Techniques

These techniques, for the most part, are reasonably secure. Unfortunately, they are let down because while authentication signatures are hidden on the disc, they still need to be read by the CD-ROM drive and because of this, it is possible to create working copies of most titles with a good CD recorder. Lite-On (amongst others) have CD recorders that are able to clone most copy protected games perfectly due to their ability to read and write sub-channel data and write discs in RAW modes (DAO RAW) using specialty cloning software. SafeDisc protection can be defeated with little effort using this method.

It is also possible to make working copies of SecuROM discs. These copies are not 1-to-1 copies but for the most part, they will work as intended on most CD/DVD drives. Software such as Alcohol 120% or Blindwrite's Blindread have a feature that can "monitor these delays and write them down in a file" (Zarathustra 2002), these delays being the sector read delays. Under Alcohol 120% this feature is called Data Positioning Measurement or DPM. These measurements are then stored in a separate file that virtual CD/DVD emulators, such as Daemon Tools, can use to present what appears to be a perfectly valid CD image to an application.

To actually create a physical copy (say to a CD-R) of a SecuROM disc, the reading delay needs to be simulated. The most common way to achieve this is to burn the same sector twice. There is no problem doing this except the sector number must also be set identical. Since there are now two physical sectors with basically the same address, the disc can no longer be considered to be technically a CD-R as it does not adhere to the standard. Most optical drives will detect the problem though and only return one sector, effectively fooling SecuROM into believing the delays are real.

While these techniques are not fool-proof for the determined hacker, they do keep honest people honest and definitely help keep piracy lower than what it would be. Furthermore, these systems can really only be applied to ISO9660 based media, so CD-ROM and DVD-ROM discs are the only medium that can benefit.

## 4.3 Application to the USB Security Device

Copy protection is even more important in security devices. There must be some method in place eventually to try to prevent unauthorized duplication of the token. Unfortunately, the three products above can only be used on optical media; they have no application to a UMSD as they measure characteristics that belong to CD's or DVD's.

Furthermore, since UMSD are designed to be quite open, there is no real way to prevent the duplication of the token. The authentication data may be hidden in the filesystem directly (Chapter 7) to obscure it, but this is still not immune to a low-level copy, such as dd under Linux.

One can employ some techniques to the actual token data to reduce the effects of duplicated however. The best method is similar to a rolling code in a vehicles keyless entry system. The remote control generates a code which seems random to anyone eavesdropping on the transmission except for the receiver in the vehicle. It is programmed with the same code algorithm as the remote and knows what code to expect next time. In fact, most systems know what to expect for the next few hundred transmissions. This is done to ensure accidental presses of the remote when away from the vehicle do not remove the numerical link between car and remote.

An access code can be embedded into the token and updated on every logon. This code would be recorded in the system's database. When the user comes back to logon, the system would check this code, see if it what should be there and if it is valid, replace the code with another. This is in essence what a single-use password sets out to achieve. If someone then duplicates the token, they must login with it before the real token is used again or else their code will be invalidated and a new code stored on the real token. In terms of achievable duplication prevention, this is probably the best that can be achieved with a UMSD. This could be supplemented with a check of the model and brand of the device from its ROM. While not foolproof, it would increase security as a hacker would need the exact make and model of UMSD to be successful.

## 4.4 Chapter Summary

This chapter discussed in detail the two biggest forms of copy protection currently available for optical media; SafeDisc and SecuROM. SafeDisc embeds a digital signature on a disc by deliberately corrupting sectors.

Early versions of SecuROM used a similar technique however from version 4.8, it switched to a different method. The time taken to read two known sectors from a disc formed the basis of the new verification system. This ment for SecuROM that every manufactured blank disc, whether CD or DVD, would be automatically deemed invalid when recorded with a SecuROM protected application.

Finally, methods of securing the token on the USB security device were evaluated. Embedding a random access code in the token and updating this on every logon is probably the best protection that can be used on a standard UMSD.

# Chapter 5

# Authentication Subsystems in Linux

This chapter discusses the authentication systems that can be used in Linux-based distributions. It then goes on to show how PAM is used to bridge the gaps in these systems and help programs use the available authentication systems effectively.

## 5.1 Current Systems Available

Currently there are not a great number of authentication systems for Linux. A short list includes but is not limited to:

- /etc/passwd file
- Shadow passwords
- LDAP
- NT Authentication
- Kerberos
- Network Information Service

The problem with all these systems is that they require the client programs to have support built in for each system. So if a developer of one program decides that shadow password authentication is all that is needed, all the users of that program are locked into using them too. What if the user wants to authenticate against an NT server or a LDAP server? They can't; either they change to a program that does authenticate against what they want or they stop using the program.

This caused lots of problems, especially with mixed operating system environments such as computer labs. User accounts may be stored on an NT server machine which was fine for the Windows computers, but not fine for the Linux or Mac ones. So then a NIS server would be setup to authenticate the Linux clients and a Mac OS server for the Mac clients. That is all well and good until users need to be added to the system; there are now three user databases that need to be updated simultaneously. To make matters worse, when a user changes their password or an account is to be disabled, again three databases need to be updated to reflect these changes. This is very tedious, especially when there are hundreds or thousands of users.

## 5.2 PAM to the Rescue

The Pluggable Authentication Module System (PAM) puts an end to all this. PAM separates the authentication methods from the client program both simplifying the program code and improving security or "to separate the development of privilege granting software from the development of secure and appropriate authentication schemes" (Morgan 2002). Program code is simplified as developers do not need to bother coding routines to interface with the different authentication systems. All that needs to be included is code to interface with the PAM application programming interfaces or API's. This means more focus can be spent on the actual program itself rather than the security of it. Security is improved as tested and trusted modules are used to handle the authentication requests instead of modules coded by the developer. This helps to ensure that simple exploits do not crop up in security-sensitive programs that may have been programmed by more inexperienced developers.

PAM works by breaking down any authentication methods and tasks into modules that can be stacked together or "pluggable". This makes PAM a very flexible authentication system. It can be used to mix and match different authentication databases together across different platforms thus allowing us to have a single user database across both Windows and Linux machines.

PAM handles four main tasks related to authentication and user management. These are:

- authentication management
- account management
- session management
- password management

Authentication handles the identification of the user, whether this is prompting them for a password, smartcard or other authentication means. It also handles the escalation of privileges through some of its own procedures. Account handles non-authentication related functions. This includes resource access, location-based and time of day access. Session is responsible for pre- and post-logon tasks, such as opening a log file, creating temporary directories or some passing of data from one program to another. Password handles "updating the authentication token associated with the user" (Morgan 2002) There is usually one Password handling module for every Authentication method specified.

Each of these module types (auth, account, session and password) has a control-flag associated with it. This lets PAM know what should happen on a module success or failure. Since PAM lets users stack many modules together, sometimes even with the same name but different parameters, it is important that the control-flags let PAM distinguish between and transfer control in a sensible matter. These module successes or failures are not passed to the application that requested authentication; rather a success or fail summary result is passed once PAM relinquishes control to the application.

There are five control flags which are supported in all versions of Linux-PAM, four

of which are module modifiers and one to aid in configuration of many PAM-aware programs. The flags are:

- required
- requisite
- sufficient
- optional
- include

Required specifies that the success of this module is required for the overall success of all the similarly named modules. It is important to note that if a module does fail, this failure will not be noticed until all the remaining modules with the same name have been executed.

Requisite is similar to required in the sense that it must succeed to succeed overall however if a module does fail, control is returned immediately to the application and the return code is from the first required or requisite module to fail. Requisite is handy when it is important to check something important related to a user's logon before they are authenticated, such as day of week for time-based restrictions or against lists of banned users.

Sufficient indicates that the success of this module is sufficient to ignore processing the other similarly named modules in the stack. It may be used when a hardware based authentication device is used on the system such as a smartcard reader. The smartcard interaction module may be given the "sufficient" flag to ensure we do not need to process the other authentication modules such as passwords.

Ignore sets a module as begin non-critical to the success or failure of the authentication request by the application. Modules marked this way are usually ignored by PAM unless all other modules have not returned definite results, such as `PAM_IGNORE`.

Include is not a module modifier, rather a syntax operator. It tells PAM to "include all lines of given type from the configuration file specified as an argument to this control"

(Morgan 2002). This allows users to create global PAM configuration files and include relevant parts of each in the application's PAM configuration.

## 5.3 Chapter Summary

This chapter discussed how PAM can be used to tie multiple authentication systems together for a single application. PAM allows application developers to focus on the features in their application rather than the security implementations. Furthermore, PAM is hugely configurable, supporting a handful of basic control flags which allow for some quite comprehensive authentication configurations. This allows system administrators to change the authentication system for any given PAM-aware application with no consequence to the performance or reliability.

# Chapter 6

# Project Methodology

This chapter outlines the different approaches that could have been taken to create each section of the device. Issues related to copy protection, encryption methods, programming languages and device interfacing are discussed.

## 6.1 Importance of Copy Protection

Research in the copy protection area will be important for this project. USB Mass Storage devices are inherently insecure due to their very open nature. This would allow for quite easy duplication of tokens onto other storage devices. There are a few methods employed by CD/DVD manufacturers, both in data and audio fields, to limit copying and redistribution of material on these mediums. Investigation into a few of the most common techniques/software would be performed. Another consideration is the support of the protection scheme under other operating systems, such as MacOS X and more importantly, due to the technical tasks in this project, Linux. This also creates further considerations; due to Linux's low level interfaces for hardware access, we may find that many of the copy protection schemes can be circumvented under it.

## 6.2 Device Considerations

The largest part of the project will be the creation of the actual software to authenticate from the USB device. This area will involve a few distinct paths of research:

- Suitable encryption methods
- Suitable programming languages
- Choice of target hardware, both computer and flash drive
- Research of authentication layers under Linux
- Preference of low level device handling

### 6.2.1 Encryption Methods

To ensure that the token/device cannot be compromised, it should have some form of encryption. There are three main approaches to achieving this:

1. Create an encryption library from scratch using a simple cipher.
2. Use a freely available industry standard encryption library, dynamically linking to it from the authentication software.
3. Use existing command line based encryption tools available for Linux.

Of the three approaches, the first is the most insecure and the most difficult. Creating a secure encryption library basically from scratch could easily be a year long project on its own and creating one as a sub-project is well out of the scope of this project. The second and third approaches make more sense in a time-restricted task. Both are viable alternatives which can achieve high levels of encryption security. The use of command line tools would lend itself to be the most flexible solution, especially since the user could substitute their preferred encryption program for the default one. This approach also has the advantage that many programs could authenticate easily off the one token

or key. Finally, the legal restrictions on encryption software in other countries are a concern. Wherever possible, software and ciphers that are available in all countries should be used in favour of encryption products with export restrictions.

### 6.2.2 Programming Languages

There are usually a handful of programming languages available to a developer that is suitable for the task at hand; C/C++, Java, Visual Basic and more recently, C#. To ensure ease of cross-platform portability, C/C++ is best suited for this project. While Java has far better cross-platform portability, its use of a Virtual Machine severely limits its direct access to hardware, which is essential to a secure authentication device. C/C++ also has the advantage of the same compiler, GNU C Complier on Linux (as gcc) and Windows (as MinGW or gcc provided through Cygwin emulation).

### 6.2.3 Target Hardware

The choice of target hardware needs to be considered, not so much for the computer, but for the choice of USB Flash Drive. The computer is assumed to be a standard 32bit x86 architecture based system with a USB controller (32 bit architecture is mentioned intentionally as some crypto functions may behave differently on a 64 bit processor). The USB drive, however, could be one of many hundreds of devices available. This can then be extended to include other devices whose primary roles are not that of a USB Mass Storage device but are secondary functions supporting a primary function, such as the storage of photos in a digital camera or the storage of MP3 and OGG files in a portable music player. For the time being, the focus will be on USB thumb drives as the primary device for the storage of the token. Another factor that could increase the security of the device is determining if some of the devices have unique (or semi-unique) serial numbers in ROM onboard, this would allow the software to lock the token to a particular hardware device and thwart most forms of token copying to other devices.

### 6.2.4 Authentication Layers

Nearly all Linux distributions have an authentication layer that keeps the authentication systems separate from the programs that request authentication and on most Linux systems, this is provided by PAM, the Pluggable Authentication Module system. When a new authentication system is added to the system, if a PAM exists for it, then any application that is PAM-aware can immediately use that new system for its authentication. PAM saves developers - of both authentication systems and software that requires authentication - a lot of time and effort; having a framework that can work on different architectures to talk to different programs is a huge advantage. One other advantage is that the quality of the code is improved due to the already secure nature of PAM; the risk of creating bugs that reduce the effectiveness of the authentication means is lower than the cost of creating the entire system from scratch.

One of the goals is to have the device working with a login manager, in our case the Gnome Display Manager or GDM. Further research must be done to determine the underlying authentication mean that GDM uses when granting access to a user. There is a good chance that it uses PAM, which if this is the case, should not require too much change to use a secondary authentication means.

### 6.2.5 Device Interface

Arguably the most difficult part of the project will be accessing the USB hardware directly. Under Linux, direct access to the USB Flash Drive as a hard drive is relatively straightforward, but accessing it as a USB device, independent of its actual role, will be more difficult. Under Linux, this could be achieved via a kernel module that sets-up a connection to the device and creates a device node in the /dev/ directory, thus controlling direct access to the device from malicious users/programs. This may be unnecessary if a suitable copy protection method can be found and successfully implemented. Furthermore, we may not need any direct access to the device; this would allow us to simply use the OS's file system layer and access the token as a simple binary file.

## 6.3 Assessment of Consequential Effects

### 6.3.1 Sustainability Issues

With anything related to computers and technology, there is always the concern that something will supersede a product. Whether it is CD writers being replaced by DVD writers, parallel ATA drives being replaced by serial ATA drives, or even floppy discs being replaced by memory cards or USB thumb drives, there is always something that will come along and replace something else. For this reason, it is important that the USB security device be forwards-compatible so users will not be disadvantaged both financially and feature based. Forward or future-compatibility is usually ensured by the use of firmware upgrades; allowing features to be added to the system at a later date without requiring the purchase of an entirely new device. The only other way is if a device is built using a technology that is unlikely to be removed from a computer system in the foreseeable future. A keyboard is a good example of this. It is quite unlikely that the keyboard will be removed from computers any time soon, or most probably not until voice recognition technology reaches a stage where it is a useful and accurate input technology. The same goes for USB. Pretty much every peripheral these days can be connected to a computer via USB. Almost every digital camera, MP3 player, printer, scanner, mouse and keyboard use USB for their interface to the computer. As a consequence of this rapid uptake of USB interfaces, it is highly unlikely that USB will be removed from computers for a very long time to come.

### 6.3.2 Safety Issues

It is important for any business who secures users data effectively that some way for the business to gain access to that data quickly is available. This may be to check on an employee's progress for a task or to recover data of a recently fired employee from their home drive. So there must be ways whereby the owner of the intellectual property, which is being protected by our system, can gain legitimate access. This may involve creating an administrator backdoor in the authentication system, for example, that when provided with the correct (and usually more secure) administrator key/token,

the administrator can access anything protected on the system. The concern here though is that now there is a key/token that can bypass all the security measures - what happens when this falls into the wrong hands (because it will eventually!)?

A related issue to this is fail-safe or backup authentication systems. Can we really provide a backup system that, while more cumbersome to use, can still provide a similar level of security that the primary means achieves? What if the fail-safe system asks the user for their logon name, password and date of birth, would that be as secure as a smartcard authentication or a USB based security device? Whatever backup system is chosen, it is very important that it be able to match the security provided by the device initially.

### 6.3.3 Ethical Issues

How can it be ensured that the system does not contain backdoors that can be exploited by the programmer at a later date to gain unauthorised access? Could the code be released under an open-source licence such as GPL. This way, anyone who had concerns about the integrity of the system could audit the code. If the system is to be marketed though, it is more likely that the code would be closed-source. While this will help to generate profits, there is no real assurance for the users who purchase the system that they are really secure. There is no way that most businesses could conduct a security audit of the binary form of the software to ensure the integrity of both it and the programmer/programming house. The system should be designed and programmed with absolutely no backdoors, for this is a big ethics violation. This does not mean override or full access codes/tokens should not be considered as many businesses would like this ability in their systems. The distinction here is that a backdoor is a hidden bypass or entry point in the system that can be exploited to gain unauthorised entry whereas an override or full access code is a documented security override that can be used by administrators of a network.

## 6.4 Chapter Summary

This chapter discussed the different approaches that could have been taken with each section of the device. Copy protection issues were investigated briefly. A number of aspects related to the device including encryption methods and programming languages were detailed. Finally a brief assessment of the consequential effects was included.

# Chapter 7

# A Hardware-Based Linux PAM Security System

This section of the project deals with the programming aspects of the security device. There are a number of components that make up the entire USB security system; these are:

- A USB Mass Storage Device
- A Pluggable Authentication Module
- The Gnome Display Manager
- AutoFS
- Utility Programs

## 7.1 USB Mass Storage Device

To begin the project, a USB Mass Storage Device (UMSD) is needed for storage of the authentication token. In all, there are no restrictions on what type of UMSD is to be used. The most sensible choice is a small USB Flash Drive or USB Thumb Drive. These devices can be purchased for under $50 in most cases and support both USB

1.1 and USB 2 specifications (*Thunderbird Computing :: Price Lists* 2005). However there is no reason why another device that shows up to a computer as a UMSD cannot be used as the storage. Devices which include UMSD support include but are by no means limited to:

- Digital Cameras
- MP3 Players
- Portable Hard Disk Drives
- PDAs

Any of these could be used as the device without any ill effects. This would help to improve the security of the system; it would not be apparent to a thief that the digital camera needs to be connected to the computer to gain access.

The other important consideration for the device is how the authentication token or credentials would be stored. Three possible methods were identified:

- Use of an encrypted token
- Use of an encrypted filesystem
- Use of hidden areas of the disk

### 7.1.1 Encrypted Token

By far the easiest method is to use an encrypted token. This also provides the system with the least protection against duplication as the token file can be copied with relative ease from one device to another. Other than that, the token is an acceptable form of credential storage. All that is needed to access the token is simple C file input/output functions such as fopen, read and write. This allows our supporting utilities to be much simpler when creating tokens and updating the system database.

### 7.1.2 Encrypted Filesystem

The use of an encrypted filesystem is a little more difficult to implement but still achievable under a Linux-based operating system. This method involves taking over the entire block device and using it as one large file. The original filesystem (usually FAT) is completely corrupted and the device cannot be used as a conventional storage device again unless it is reformatted. Note the word "conventional"; it is possible to provide some crypto-loopback storage on the device once the entire disk is encrypted. The last 100 MB of a 128 MB drive may be a disk image that once the user is authenticated by the system, it is decrypted in memory and the user can access it from a designated mount point. Once the user logs out, the disk is unmounted from memory, re-encrypted and copied back to the device. This would give the user secure storage for the transporting of sensitive documents and could be implemented seamlessly with the PAM so zero user interaction is required.

### 7.1.3 Hidden Storage Areas

The use of hidden areas of the disk is by far the most difficult to implement but would return the greatest payoff. If implemented correctly, there would be no visible evidence that the device contains authentication credentials. In this method, authentication data is stored in unused areas of the disk, areas such as the master boot record (MBR), boot sector of the filesystem and other hidden or reserved areas of the filesystem.

The master boot record is arguably the easiest of the three methods. This is space reserved at the start of the disk as part of the partition table where usually a boot loader is placed. There should be little or no impact by placing a few bytes of data here. The only concern is if the computer is booted with the device connected and the system is capable of USB booting. The system may think the device is bootable and start loading the MBR into memory. This is quite undesirable, especially if the data loaded does correspond to some CPU instructions and the computer actually executes something instead of hanging. One way around this may be to take a standard Linux boot loader, say LILO (LInux LOader), customize it to pass the boot loading to the first hard disc drive, install it on the device and place our authentication data just after

it.

Hiding data in the boot sector of a filesystem is a similar task to hiding it in the MBR. The only difference is that the MBR is queried when the system is booting if specified in the boot sequence whereas the boot sector of a partition can only be queried if asked directly by a boot loader or failing to query a MBR, the first boot sector will be activated if it contains bootable code. Other than that, once we have the start of the boot sector we can access it as if we were accessing a MBR.

There is one big problem when programs start playing in the MBR; anit-virus programs. If anything other than the operating system (and sometimes the OS is not exempt from the restrictions too!) attempts to access the MBR of any disc on a system, the anti-virus software will detect this as a boot-sector virus attempting to replicate to other drives and will deny access. Since we are using Linux for the time being, this is of no real concern except if the device is used in Windows as well. If the anti-virus program uses a technique known as heuristics to help detect new and unknown viruses, it may decide that the changes made to the MBR of the device are in line with a boot-sector virus and will alert the user accordingly.

Linux filesystems such as Ext2 or Ext3 (Linux 2nd and 3rd Extended Filesystems) store what is known as a superblock backup several times throughout a formatted filesystem. Since these are all backup copies, there should be no need to use them in the general operation of the filesystem. One of these superblocks could be erased and authentication data placed in its space. This would result in exceptionally good data hiding which is immune to nearly all duplication processes (except for dd). On paper this sounds like a great solution except that the operating system updates all the superblocks after a change to the master superblock and as a result, the authentication data would be constantly lost. The only way this method would work was if it is possible to allocate the superblocks on a device and then remove one of them from the list of superblocks without re-allocating the space to the filesystem. This is probably not achievable as the location of superblocks is not usually stored as a list of known locations but rather a formula that is evaluated with respect to the size of the device. By far an easier method may be to simply mark with software bad sectors on the device. This has the benefits of never being overwritten by the filesystem as it is a (semi) permanent no-go

barrier.

## 7.2 A Pluggable Authentication Module

A PAM must be created to interface between the client applications (GDM) and the authentication device. This module handles the following tasks:

- Loading the configuration file
- Verifying that the user exists on the system
- Checking that a no-logon file exists
- Opening the token off the device and performing an MD5 sum
- Comparing this calculated MD5 sum to the one stored in the database

To speed up the development of the PAM, `the pam_permit` module was used as a starting framework. The sole purpose of this module is to simply return `PAM_SUCCESS` whenever called. While it does have all four PAM sections defined (authentication, account, session and password), it only has logic under the authentication management functions. This code simply checks the user exists on the system then permits them while all the other sections blindly return a `PAM_SUCCESS` code. There is no problem with this implementation as the definition of the module is indeed to always return success. This module was chosen to build upon because the simplicity of its code and layout makes initial testing of basic functions easier than programming a full module from scratch.

Upon execution, the module loads the default configuration file, namely `/etc/usbsecure.conf`. This file contains the path to the token database and to the mount point for the UMSD. Also the path to the no-logon or deny file is read at this time. This configuration is stored in a binary format, not plain ASCII text. A configuration structure is created using the supporting utilities and the actual structure is stored in the file directly.

Listing 7.1: `usb_config_struct` Configuration Structure.

```
#ifndef _CONFIG_H
#define _CONFIG_H

#define CONFFILE "/mnt/homes/matthew/uni/project/etc/usbsecure.conf"

struct usb_config_struct {
        char tokenfile[4096];
        char denyfile[4096];
        char dbfile[4096];
};

#endif
```

Next the module sets out to verify that the user exists. The system does not want to attempt to login someone who does not exist, especially if the system allows these logins to occur and passes the root filesystem as the home directory. The call to `pam_get_user` handles this. If the user is unknown to the system, a `PAM_USER_UNKNOWN` value is returned and the module exits otherwise a `PAM_SUCCESS` is returned and the authentication can continue.

After the username is validated by the system, the module checks for a deny file. This file is similar to the `/etc/nologon` file which prevents users from logging into a system, especially useful when performing system maintenance on a live server. If this file exists and it contains a 'k' character (meaning ok to login) then authentication can continue as normal. If any other character is found a `PAM_AUTH_ERR` is returned. This deny feature is important in situations where a token may be compromised and it is unknown which one exactly. The administrator can activate the deny file and prevent all token logons until the situation is resolved. In this time, users will be simply forced to use a backup authentication system, most probably a password.

Once the deny file condition has been passed, the module can begin reading the token. To begin with the module loads the token database and attempts to locate the token signature for the previously entered username. Even though the user is known to exist

on the system at this point, the search can still fail if the user has not been granted a corresponding token. If it is not found, a `PAM_AUTH_ERR` is returned. The token signature is then passed along with the token path to the `do_md5_file` function. This function takes a file to compute the MD5 hash from and the expected MD5 hash and returns 1 if they do not match and 0 if they do. If one is returned, the module once again returns `PAM_AUTH_ERR` otherwise the success message is passed to the application using the PAM conversation-¿conv function and a `PAM_SUCCESS` is returned.

All the MD5 functions are supplied by external code found on the internet. Writing a message digest algorithm and the resulting functions is not an easy task from scratch. Most implementations rely on either transformation tables, bitwise binary math or even both in some cases. It was decided that time would be better spent integrating different parts of the authentication system together rather than attempting to write a MD5 library.

## 7.3 The Gnome Display Manager

The Gnome Display (or Desktop in recent times) Manager (GDM) is one of the most widely used graphical login manager for Linux distributions. On system boot-up, it starts the Xorg or XFree86 Windowing System and presents the user with a login screen. GDM is a logical choice for this project as it is a PAM-aware application. This means it requires no changes at all to the source code to add and manipulate authentication methods. GDM can also be reconfigured to use what is known as a "face browser"; this is similar to the welcome screen on a Windows XP machine with the icons or "faces" with each user listed next to the corresponding picture. With the face browser configured correctly, all a user needs to do is to connect the UMSD and double click on their face to login.

Configuring the PAM's for GDM is a relatively straightforward task. Under Gentoo Linux, application-specific PAM configurations are found in the `/etc/pam.d/` folder. Each application then had a configuration file which corresponds with its name that contains the PAM options. The required PAM configuration for GDM in this project

is shown below:

Listing 7.2: PAM configuration for GDM with USBSecure.

```
#%PAM-1.0
auth       sufficient   /lib/security/pam_usbsec.so
session    required     /lib/security/pam_motd.so
auth       required     /lib/security/pam_env.so
auth       required     /lib/security/pam_stack.so service=system-auth
auth       required     /lib/security/pam_nologin.so
account    required     /lib/security/pam_stack.so service=system-auth
password   required     /lib/security/pam_stack.so service=system-auth
session    required     /lib/security/pam_stack.so service=system-auth
session    optional     /lib/security/pam_console.so
```

It can be seen in the code above that the `pam_usbsec` module is an authentication (auth) module. Furthermore it is marked as being sufficient; as discussed earlier, if this module succeeds then PAM is satisfied that the user is authenticated.

## 7.4 AutoFS

AutoFS is not a necessary component of the final system. This set of tools automatically mounts and unmounts filesystems on demand without any user interaction. This was ideal for the UMSD initially as when the PAM would attempt to access the token the device would be auto-mounted. It was decided not to use it in the end due to issues with some computers and the need for a kernel module. Some systems would not auto-mount properly, failing to initialise on boot and thus failing to auto-mount any filesystems. Secondly this implementation requires a kernel auto-mount module to either be loaded or compiled into the kernel. While all 2.6 series Linux kernels feature this module code, it was undesirable to require kernel level modifications to utilize the device.

Since AutoFS was not to be used, there was still a need to mount the device prior to authentication and to unmount it immediately after. This was accomplished by simply

mounting the device manually using a mount call from the PAM before the token was to be accessed and unmounting manually just after the token signature was computed. This was deemed to be acceptable as access to the device is enabled when needed with minimal effort.

## 7.5 Utility Programs

A small number of utility programs were created to handle configuring, administering and querying the system. Currently there are three programs:

- mkconfig
- addtoken
- viewdb

### 7.5.1 Mkconfig

Mkconfig is used during the initial setup phase to create the configuration file for the PAM. It creates a `usb_config_struct` structure and sets the token path, deny file path and token database path and writes the structure to the configuration file as a binary object.

### 7.5.2 Addtoken

Addtoken creates an entry in the token database and adds the user and token signature to it. A `db_struct` structure is created to hold the user information in the token database. The definition of the `db_struct` object is shown below:

Listing 7.3: `db_struct` header definition.

```
struct db_struct {
        char username[128];
        unsigned char tokenmd5[33];
```

};

Once the structure is created the configuration file is read, followed by the hashing of the new token. Next the user is prompted for the username to be associated with this token. It is critical that this username corresponds with one on the system and that it is entered exactly the same as in the system; this means identical case. Finally the structure is appended to the end of the token database as a raw binary dump of the structure data.

### 7.5.3 Viewdb

Viewdb is a simple utility program to view the contents of the token database. It outputs every token found in the database, detailing username and token hash. It also has the ability to check if a token is currently connected and if so, checks who the token belongs to by indicating on the output if a token is currently attached.

Each of these utilities and the PAM itself all use a shared set of header files. One named db.h contains the above `db_struct` definition (Listing 7.3) and one named conf.h that contains the usb_config_struct definition (Listing 7.1). These are used to ensure that all functions and programs that require access to these interfaces are always using the correct version as changing a header file means all dependencies are updated. This is very important, especially if third party programs are used one day to modify the token database. The program can simply include these files and it will have correct definitions of all the structures. If the author simply copies this code directly into their program, changes to the headers will not be propagated to their software and thus as a result may cause severe problems or even data corruption.

## 7.6 An Analysis of the Current Implementation

The current implementation of the security system does have some limitations and security concerns. Many of these are beyond the scope or control of this project but still are important to consider. The main issue here is if a hacker gains control of the

root account on the system. If this occurs, there is zero security left on the system. The token database, while protected by unauthorised changes through the use of filesystem permissions, cannot be secured effectively against any root user[1].

Root access can be made quite easy for a hacker to gain if a few simple checks or options are not disabled or reconfigured from default settings or if simple procedures are not in place. These include:

- Setting a BIOS/CMOS password
- Changing the default boot sequence
- Securing the default Linux boot-loader
- Physically securing the system

### 7.6.1 BIOS/CMOS Passwords

Having a BIOS or CMOS password set is one of the most basic and most important steps in securing a system. While these passwords are easy to reset if a hacker can gain physical access to the systems mainboard, they are the first line of defense in protecting or enforcing the other options. If the default boot sequence is secured but no password is set on BIOS, then the hacker can simply enter, change the boot sequence to something that favours them and reboot.

### 7.6.2 Boot Sequences

"If a hacker can boot off an external drive, then it's not your system anymore" (*Microsoft TechNet: 10 Immutable Laws of Security* 2005). This is a very real concern, especially with all the latest Linux Live-CD which allows you to boot up a full-blown Linux system and access any filesystems on the machine, completely ignoring all filesystem permissions. If a hacker can boot off a CD, floppy disk or even a UMSD, they can

[1]The exception to this rule is the immutable permission. When enabled, not even root can change the standard file permissions (except of course root can remove the immutable permission!)

change boot-loader settings, create false accounts on the system or in this case, tamper with the USB secure token database.

### 7.6.3 Boot-Loaders

Linux boot-loaders are often overlooked by most people as an entry point for hackers. By default nearly every distribution installs its boot-loader (whether this is LILO or GRUB) without any password. Many do give the option in setup to add a password, however the option is usually small and has little or no emphasis drawn to it. By simply appending the keyword “single” to the kernel options line in the boot-loader, nearly all Linux distributions will be booted into what is known as ”single user mode”. In this mode, root access is granted to the console immediately thus allowing a hacker to bypass all security measures with great ease. Furthermore, a hacker could instruct the boot-loader to boot some other foreign media, such as a CD, regardless of the BIOS boot sequence option. This is a simple hole to plug, enabling the password option on the boot-loader when installing will completely prevent this exploit on any system.

### 7.6.4 Physical Security

Without any physical security on a system, there is absolutely no point enforcing the other points above. It is important to note that the amount of physical security is directly proportional to the sensitivity of the data that needs to be protected. So for a home user, simply placing a small padlock on the side of the case is ample. A paranoid user may lock the case in a small cupboard next to the desk. If a business is to take physical security seriously, servers should be locked in racks which are in turn locked in a server room. By far the most secure server is one which is powered off, unplugged and placed in a locked room. This is quite an extreme case, especially because the server cannot actually do anything then other than take up storage space but the point is still valid. A secure server is one which has its physical access restricted to authorised personnel, is constantly patched with all the latest security updates and is continuously monitored for any suspicion activity.

The last point there is one that is overlooked all too many times in industry. Many system administrators will simply secure their servers in a room and ensure all the latest security patches are applied as they are released and that's it. They do not monitor the server logs, check the event viewer, browse the filesystem or even check the running process list for anything that looks out of the ordinary. For a system to be considered secure, it must be monitored regularly by some personnel who have reasonably good knowledge of the system.

## 7.7 Progress

The device and supporting software is at a usable state. There have been no major authentication problems in the initial testing phase. Issues were found with the AutoFS daemon on one of the test machines. Once installed and the devices defined, AutoFS would simply fail to load, either on system start-up or when manually restarted. Since AutoFS relies on kernel code to perform access detection and mount/unmount the filesystem accordingly, the kernel of the problematic machine was inspected and found to be fine. To be safe, the kernel was recompiled again with no change in the reported errors.

Another bug was with the PAM conversation functions. These functions allow developers to pass messages from the PAM to the calling application, such as "Please enter password" or "Swipe access card now". Two conversations were initially coded into the module; one for a successful login and one for a failed login. On successful logins, the message was displayed correctly however failed logins initially displayed the correct message but as the module progressed, it began to fail.

In worst cases this fault would cause GDM to restart rather than simply exiting and letting the secondary login system work. This was remedied by simply commenting out the conversation code for failures. It is unknown why this problem has occurred. Further investigation by running GDM through a debugger such as gdb and monitoring the segmentation faults and/or exceptions is probably the only way to resolve this issue.

The final device does not perform any encryption on the token. It was hoped to use

Gnu Privacy Guard (GPG) to encrypt the token so it could not be tampered with. After researching control of GPG by a third-party application (namely the PAM) it was found that the GPG Made Easy (GPGME) library was the best system for the job. GPGME is an API that is suppose to make interfacing with a GPG installation relatively straightforward. Anything that needs to be done can be coded into an application directly and executed by the library at runtime. Unfortunately, GPGME was quite difficult to grasp. It does come with many examples but these either did not work as expected or were too complicated to reproduce in the PAM. In the end it was decided to drop the encryption of the token and use a much simpler Message Digest 5 (MD5) hash of the token file.

While it may seem like not encrypting the token has significantly reduced the security this device can provide, it should be noted that as long as the token is kept reasonably secure, the security is no different to that of an encrypted token since the hacker cannot obtain a copy of the token file. A simple example, most people would not give their car keys or their credit card to someone they do not trust. If the same logic is applied to the UMSD then it is obvious that there is no loss in system security. Sure if a hacker obtains the device they could duplicate it, but it is also the same as giving a set of car keys to a valet; they could duplicate the key and then have full access to that vehicle!

## 7.8 Chapter Summary

This chapter covered all the required software components for the project. A detailed overview of the PAM was shown, including file definitions for important data structures. The Gnome Display Manager is the application that was tested to use the USB security device. A sample PAM configuration for GDM was also shown, examplifying how simple configuring new authentication methods for existing PAM-aware applications can be.

The software also has a handful of supporting utility programs to perform system maintenance and gather information for administrators. Three programs were developed; mkconfig, addtoken and viewdb. These allow administrators to create a default usbsecure.conf configuration file, add a new token to the database and view the known tokens in the database.

Finally it was shown that there are some ways to get around the security provided by the device. All of these are common sense issues though and thus should already be taken care of if the systems are administered by competent IT staff.

# Chapter 8

# Extending The Security Device

This chapter outlines enhancing the device to operate in a networked environment, similar to ones present in large enterprises.

## 8.1 Network/Distributed Authentication

Without some form of network authentication, the potential applications for the device are quite limited. An enterprise would definitely not consider a system that could not be deployed and managed over a network to a large number of computers. There is no way they would implement an authentication system whereby each machine has its own list of users to authenticate to, it would become unwieldy very quickly!

There are two approaches that could be taken to implement a network-enabled version of the device:

- A fully networked system using a single authentication server to process all authentication requests.
- A fully networked system using a local database for authentication, but with the ability to automatically import keys from surrounding computers.

## 8.2 Client/Server Approach

The first approach is typical of most network systems currently available. NT Domain Authentication is a prime example. For a workstation to be logged onto the network it must be validated by a domain logon server or else it is denied access to any resources on the network. Appling the same idea to the device, the local token database would be moved to a central server and the direct file access commands replaced with networking code.

Storing all the tokens on a central server is a good choice from an administration standpoint. There is only one machine that needs to be secured well, after all the hackers will have to target the server now to compromise the token database. Furthermore, revoking a token is easy as there is only one system that needs updating, there is no need to iterate through all the machines in a business, searching for a particular token and revoking it when found.

## 8.3 Distributed/Peer to Peer Approach

The second approach is a much more elegant one. In this design there does not need to be a central token server, machines can send authentication requests to one-another. Each machine maintains a local token database as per the normal setup however there is one fundamental change; if a device is connected and it is not found in the local database, the machine can send a request to other machines for the token.

In this scenario a web of trust is created. A machine will trust itself and one or more other machines. If a machine cannot handle a request, it is allowed to pass it up the web a predetermined number of times. The beauty of this system means that a central token server can still be sitting at the top of the web but would not usually be queried on every logon as some machine below would have the desired token in its database.

A typical logon session may go something like this:

1. A user connects their token device to a workstation.

2. The system checks the token and cannot locate it in its local database.

3. The system then sends a request to the first machine from its trust list and asks whether it has the token in its database.

4. If the queried machine has the token and it is valid, it sends a validation response to the user's workstation. If the machine does not know the token, it sends an unknown response to the user's workstation and optionally can inform the user's workstation of the computers it trusts to aid the user's workstation in locating the token. This step (4) is then repeated.

5. The user's workstation then adds the token to its database automatically, however it sets a remote trust flag that signifies that the token is trusted from a remote source.

6. The user is then logged in as if their token was in the local database the entire time.

In step 5 it is mentioned that a remote trust flag is set to signify the token is remotely trusted. This is done so automatic revocation can be performed. If at a later date a user's token is revoked or deleted, it will be removed from the system with the initial local copy however any system that logged the same user in remotely from the trust this machine provided will still allow the user to login. With the remote trust flags in place, the system will realise when the user connects the token that it is remotely trusted and will then send a validation request to the machine the provided that trust initially to see if the token is still valid. The remote machine will then return a token revoked message and the system will deny the user access to the system and remove the token from the local database.

Another important use for the remote trust flag is to ensure that if someone asks a machine to check for a token, it must not return a validation response unless the token is trusted locally. If it remotely trusts a token it should return a remotely trusted message with the name of the system that locally trusts that particular token. That way the system requesting validation can ask the machine with the token directly.

The other important consideration for the distributed method is how to exchange data

from one machine to another. A possible solution is to use the Secure Shell (SSH) protocol with host keys. Each machine runs an SSH server and has an associated host key. For a machine to trust another, it must have the remote machines host key stored in its local `.authorized_keys` file. This will then allow the machine to connect securely and seamlessly to the other machine with no other required information.

Data can then be passed to and from each machine. Commands such as "Do you have this token?" or data such as the MD5 hash for a token may be passed with little effort. Furthermore, this approach would allow trusted computers to not only be on the local network, but anywhere across the internet. Since all communications using this method are secured quite well with SSH, there is no worry as to the integrity of the data in transit from one machine to another.

## 8.4 Chapter Summary

This chapter has described the basic framework for distributed authentication system that could enable the use of this security device in a networked environment. A conventional client/server architecture was briefly touched on before a detailed description of a peer to peer based authentication system was given. This system works by creating a "web of trust" between the computers (and if required, servers too) on the network.

# Chapter 9

# Conclusions and Further Work

This chapter concludes the dissertation. The objectives stated at the beginning are compared with the results achieved on the completion of the project. Furthermore, a brief discussion of further work that may be performed in the future that may improve the quality of the system.

## 9.1 Achievement of Project Objectives

The following objectives have been addressed:

### 9.1.1 Researching

*1. Research the current state of authentication systems for both Linux and Windows operating systems.*
*2. Research the benefits of such a device in the public domain, including social and economic issues of all involved parties.*
*3. Research cryptography and copy protection methods implemented on various media and security devices.*

Chapter 3 detailed the current authentication systems used in Linux-based operating systems. The specification also stated that research was to be conducted for the Windows operating system however this was not completed fully. A broad overview of security devices available was conducted, covering both Smart Cards and Biometric Authentication Devices. Two commercially available authentication systems were also compared; Cryptocard and Dekart Logon. Cryptocard supports a wider range of operating systems, but is more expensive than the Windows-only Dekart Logon.

The benefits gained by such a hardware-based security device was most evidently portrayed in Chapter 2. If hackers need a physical device to gain access to a system then Social Engineering attacks will drop significantly. The characteristics of a good Social Engineer were also discussed. The purpose of these qualities are to help hackers extract information from their victims efficiently but these of course would be rendered useless if a hacker was up against a two-factor authentication system.

Protection against token/device duplication is an important aspect of a security system. Chapter 4 described how some of the most common forms of copy protection for optical media are achieved. It then shows how these protections can then be circumvented with common software packages available on the internet. Since a UMSD is quite open, protection against duplication must be done within the token possibly by using a single-use access code and embedding this into the token after every logon.

### 9.1.2 Design and Implementation

*4. Using a USB flash device as the underlying hardware, design and implement a security device under a Linux-based operating system.*

The security device was created using a number of software components plus a USB flash drive. These software components comprise of a PAM, GDM and three utility programs. The PAM was developed using an existing module for the inital framework and then expanded with further code to load a configuration file, open a token file and compute a MD5 hash and then compare this hash to one from a local database. If the hash corresponds with the correct entry in the database a success message is returned

to GDM and the user is logged into their account without any further interaction. The addition of token signatures to the database is handled by one of the supporting utility programs. The other two programs allow administrators to view the token database and create an initial configuration file.

Since GDM is already a PAM-aware application, no modifications to the code were required to fully utilise the module. GDM successfully logged the test user in every the time. GDM was also configured to drop back to a password box on an unsuccessfull USB logon.

### 9.1.3 Evaluation of the Device

*5. Analyse the performance of the device in terms of robustness, security and error-recovery.*

There were some initial debugging issues that caused numerous errors, mainly with PAM conversations. After these setbacks, the authentication performance has been exceptional. When a user clicks OK there is no noticeable delay while the PAM checks the token. The other programming issues encountered were with the GPG Made Easy Library. It was not successfully integrated into the USB security system and as a result, the token has no encryption. This is not a grave concern as outlined in Chapter 7.

## 9.2 Further Objectives

*6. Research the issues related to the implementation and porting of the security device software on other Operating Systems including Windows NT based OS and Mac OS X.*

Since this device is designed to work on Linux-based systems only, there have been discussions arising throughout the project as to the feasibility of porting the security device to other operating systems, namely Microsoft Windows and Mac OS. In porting, there are a few areas that must be considered and evaluated thoroughly to ensure the functionality and integrity of the device. These are:

- Integration with authentication subsystems
- Choice of programming language and compiler
- Availability of system documentation
- Ability to port code inline or fork code-base

By far the biggest challenge is interfacing with the authentication system built into the operating system. Under Windows this is difficult to achieve at best of times; due in part to the closed-source nature of Windows. Microsoft does not release the source code for its OS's (or nearly anything they produce for that matter) and this makes developing software to mesh with their products hard to achieve. They do have a comprehensive Application Programming Interface (API) for their products, but these still do not give the developer any indication how the system is performing those functions. This makes low-level Windows programming slow and complex. Furthermore, a driver developer kit is usually required to develop device drivers with any hope of success.

These obstacles are greatly reduced under Mac OS, Mac OS X in particular. This OS is build upon a BSD base, named Darwin in Apple's case and as a result gives the OS a Unix-like feel especially when programming. Furthermore, Apple has been releasing the Darwin portion of Mac OS X to the public for free, making it easier for developers to get their hands on a copy. Darwin does not contain any graphical components of Mac OS X and as a result cannot be used solely as the development platform although it is a good place to start for low-level programming tasks such as device access.

*7. Design and implement a basic client/server system allowing the device to support roaming amongst different systems.*

Chapter 8 outlines two approaches to network-enabling the security device. The first method involves a conventional client/server relationship. This may simply be a central token server or something more highly integrated, such as a Microsoft Windows Server with an Active Directory syncronised with the token signatures.

The second method involves a distributed approach. Each machine trusts at least one other machine for authentication information. When a user wishes to logon, if the

machine does not know the user from its local database, it can send a request to any machine it trusts in the hope that one will trust this user. This method could be secured by using SSH to interface between machines.

## 9.3 Further Work

This project is currently in the infancy stage; there are many improvements that can be made. These include but are by no means restricted to:

- Addition of a Graphical User Interface (GUI) for the management of the security devices features.
- Encryption of the token.
- Addition of a single-use code to the token.

It would also be favourable to have the code inspected by another developer (or two). The code is still messy at best in some parts. Any review of the code would ensure that any design flaws or pitfalls that the initial developer may have missed would be corrected in future revisions.

# References

*CD Media World - CD/DVD Protections - CD/DVD Copy Protections - CD/DVD* (2005), CD Media World.
`http://www.cdmediaworld.com/hardware/cdrom/cd_protections.shtml`
current May 2005.

*Cisco Self-Defending Network - Cisco Systems* (2005), Cisco Systems, Inc.
`http://www.cisco.com/en/US/netsol/ns340/ns394/ns171/ns413/networking_solutions_package.html`
current July 2005.

*Conlab Pty Ltd* (2005), Conlab Pty Ltd, Doncaster East, Victoria.
`http://www.conlab.com.au`
current May 2005.

Cringely, R. X. (2005), *I, Cringely . December 20, 2001 - Good Morning, Osama — PBS*, PBS.org.
`http://www.pbs.org/cringely/pulpit/pulpit20011220.html`
current April 2005.

*Cross Match: Introduction to Biometrics* (2005), Cross Match Technologies Inc, USA.
`http://www.crossmatch.com/biometrics_intro.html`
current April 2005.

*CRYPTOCard Corp* (2005), CRYPTOCard Corp, Canada.
`http://www.cryptocard.com`
current May 2005.

*Free download - secure login for Windows - USB flash disk/smart card, biometric login with Dekart Logon* (2005), Dekart, Republic of Moldova.
`http://www.dekart.com/products/authentication_access/logon/`
current April 2005.

Grammeman, S. (2004), *Would you trade your password for chocolate?*, The Register.
`http://www.theregister.co.uk/2004/05/28/password_advice`
current April 2005.

Granger, S. (2001), *Social Engineering Fundamentals, Part I: Hacker Tactics*, SecurityFocus.
`http://www.securityfocus.com/infocus/1527`
current April 2005.

IIett, D. (2005), *Linux targeted with two-factor authentication*, ZDNet UK.
`http://news.zdnet.co.uk/internet/security/0,39020375,39196891,00.htm`
current May 2005.

Kay, R. (2005), *Biometric Authenticaton*, Vol. 39, Computerworld.
EBSCOhost database Academic Search Premier
current April 2005.

Leyden, J. (2003), *Office workers give away passwords for a cheap pen*, The Register.
`http://www.theregister.co.uk/2003/04/18/office_workers_give_away_passwords`
current April 2005.

*Microsoft Mouse and Keyboard Hardware - Fingerprint Reader Products* (2005), Microsoft Corporation, USA.
`http://www.microsoft.com/hardware/mouseandkeyboard/productlist.aspx?fprint=yes`
current May 2005.

*Microsoft TechNet: 10 Immutable Laws of Security* (2005), Microsoft Corporation.
`http://www.microsoft.com/technet/archive/community/columns/`

security/essays/10imlaws.mspx

current April 2005.

*Microsoft TechNet; Security Administration: 10 Immutable Laws* (2005), Microsoft Corporation.

http://www.microsoft.com/technet/archive/community/columns/security/essays/10salaws.mspx

current April 2005.

Morgan, A. (2002), *The Linux-PAM System Administrators' Guide*, Kernel.org.

http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html/pam.html

current April 2005.

*SafeDisc FAQs* (2002), Wine HQ.

http://www.winehq.com/hypermail/wine-devel/2002/04/att-0616/01-safedisc.txt

current September 2005.

Shinder, D. (2005), *How to Defend your Network Against Social Engineers*, WindowSecurity.com.

http://www.windowsecurity.com/articles/Social_Engineers.html

current April 2005.

*Smart Cards* (n.d.), Microsoft Corporation.

http://www.microsoft.com/resources/documentation/Windows/XP/all/reskit/en-us/Default.asp?url=/resources/documentation/Windows/XP/all/reskit/en-us/prdp_log_akio.asp

current April 2005.

*Social engineering (computer security) - Wikipedia, the free encyclopedia* (2005), Wikipedia Foundation.

http://en.wikipedia.org/wiki/Social_engineering_%28computer_security%29

current April 2005.

Taschek, J. (2001), *An Eye on Biometrics*, EWeek.com.

http://www.eweek.com/article2/0,1759,45623,00.asp

current April 2005.

*Thunderbird Computing :: Price Lists* (2005), Thunderbird Computing.

http://www.thunderbirdcomputing.com.au/prices.php?do=memory

current October 2005.

Zarathustra (2002), *Alcohol Support Forum -¿ Some facts about SecuROM v4.8x*, Alcohol Soft.

http://forum.alcohol-software.com/index.php?act=ST\&f=32\&t=1395

current September 2005.

# Appendix A

# Project Specification

University of Southern Queensland
Faculty of Engineering and Surveying

# ENG 4111/4112 Research Project
# PROJECT SPECIFICATION

| | |
|---|---|
| **For:** | Matthew Quarisa |
| **Topic:** | USB Based Security Device |
| **Supervisor:** | Dr John Leis |
| **Sponsorship:** | Faculty of Engineering and Surveying, USQ |

**Project Aim:** The aim of this project is to implement an authentication system for computers using off the shelf hardware, such as a USB Flash Disk. It also endeavors to investigate and discuss other security devices available as well as the security measures and protocols that can be used with devices today.

**Programme:** **Issue A, 24th March 2005**

1. Research the current state of authentication systems for both the Linux and Windows operating systems.
2. Research the benefits of such a device in the public domain, including social and economic issues of all involved parties.
3. Research cryptography and copy protection methods implemented on various media and security devices.
4. Using a USB flash device as the underlying hardware, design and implement a security device under a Linux-based operating system.
5. Analyse the performance of the device in terms of robustness, security and error-recovery.

As time permits:

6. Research the issues related to the implementation and porting of the security device software on other Operating Systems including Windows NT based OS and MacOS X.
7. Design and implement a basic client/server system allowing the device to support roaming amongst different systems.

**AGREED:**

| _____ | _____ | ___ / ___ / ___ |
|---|---|---|
| *(student)* | *(supervisor)* | *(dated)* |

# Appendix B

# Screenshots of the Logon Process

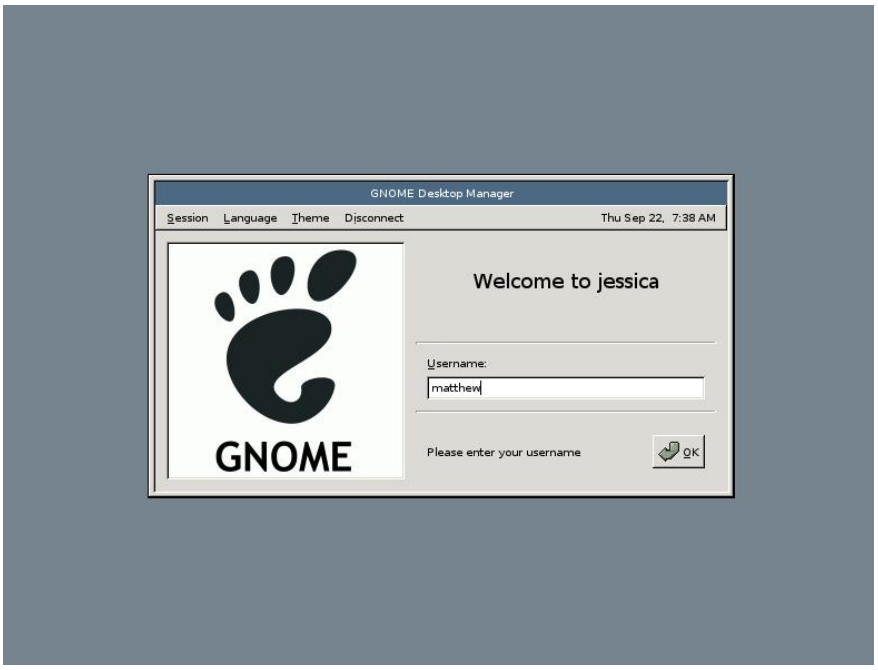


Figure B.1: First Stage of Logon

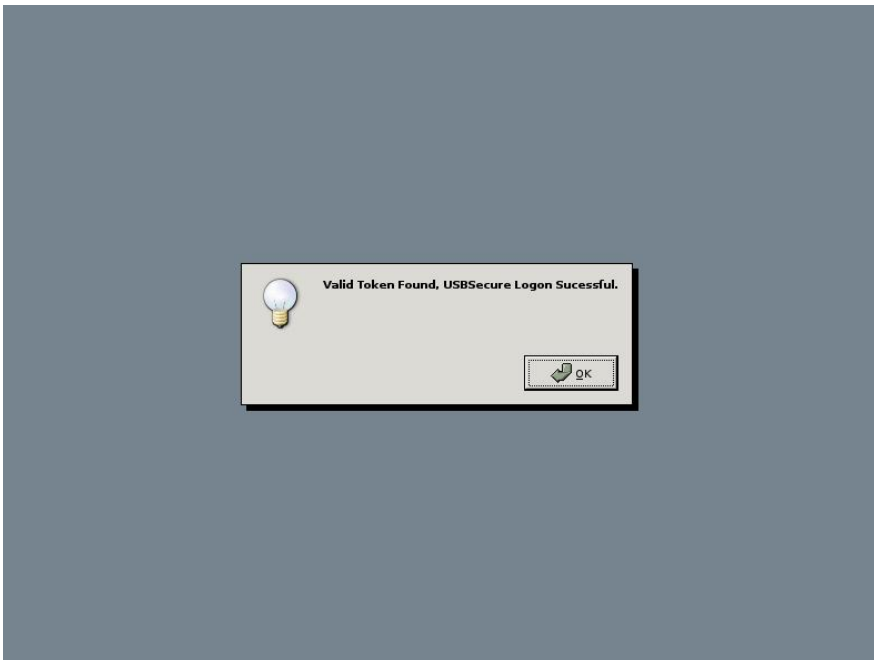


Figure B.2: Successful Logon

# Appendix C

# Source Code Listings

## C.1 Header Files

Listing C.1: conf.h code listing.

```
// conf.h - Configuration Header
// Matthew Quarisa - Q11215969

#ifndef _CONFIG_H
#define _CONFIG_H

// default config file path for testing purposes
#define CONFFILE "/mnt/homes/matthew/uni/project/etc/usbsecure.conf"

struct usb_config_struct {
        char tokenfile[4096];
        char denyfile[4096];
        char dbfile[4096];
};

#endif
```

Listing C.2: db.h code listing.

```
// db.h - Token database header
// Matthew Quarisa - Q11215969

#ifndef _DB_H
#define _DB_H

struct db_struct {
        char username[128];
        unsigned char tokenmd5[33];
};

#endif
```

Listing C.3: md5.h code listing.

```
// Part of the MD5.C library

#ifndef _MD5_H
#define _MD5_H

#ifndef uint8
#define uint8  unsigned char
#endif

#ifndef uint32
#define uint32 unsigned long int
#endif


typedef struct
{
    uint32 total[2];
    uint32 state[4];
    uint8 buffer[64];
}
md5_context;
```

```
int do_md5_file( char *file, unsigned char *thesum);
void md5_starts( md5_context *ctx );
void md5_update( md5_context *ctx, uint8 *input, uint32 length );
void md5_finish( md5_context *ctx, uint8 digest[16] );

#endif /* md5.h */
```

## C.2 Program Files

Listing C.4: pam_usbsec.c code listing.

```
/* pam_usbsec module */

/*
 * $Id: pam_permit.c,v 1.3 2004/09/22 09:37:49 kukuk Exp $
 *
 * Written by Andrew Morgan <morgan@parc.power.net> 1996/3/11
 *
 * Now pam_usbsec, by Matthew Quarisa - Q11215969
 *
 */

#define DEFAULT_USER "nobody"

#include <stdio.h>
#include <stdlib.h>
#include <sys/stat.h>
#include <string.h>

/*
 * here, we make definitions for the externally accessible functions
 * in this file (these definitions are required for static modules
 * but strongly encouraged generally) they are used to instruct the
 * modules include file to define their prototypes.
 */

#define PAM_SM_AUTH
#define PAM_SM_ACCOUNT
#define PAM_SM_SESSION
#define PAM_SM_PASSWORD

#include <security/pam_modules.h>
#include <security/_pam_macros.h>

// our includes
#include "md5.h"
#include "conf.h"
#include "db.h"

/* --- authentication management functions --- */

PAM_EXTERN
int pam_sm_authenticate(pam_handle_t *pamh,int flags,int argc,const char **argv)
{
        int retval;
        const char *user=NULL;
        struct pam_conv *conversation;
        struct pam_message message;
        struct pam_message *pmessage = &message;
        struct pam_response *resp = NULL;
```

```
struct stat st;

FILE* logfile = fopen("/tmp/LOG.usbsec", "a");
fputs("Begin logging.", logfile);
fputs("\n", logfile);

//
*****************************************************************************
// here we read the config file, it is a binary one with a config struct
struct usb_config_struct config;
FILE *conffile = fopen(CONFFILE, "rb");

if( (fread( &config, 1, sizeof( struct usb_config_struct ), conffile ) ) < 1 )
{
    // error reading file
    fclose(conffile);
    return PAM_AUTH_ERR;

}
//fclose(conffile);

//
*****************************************************************************
// put these messages in a file
char tellthem[] = "USBsecure is active but an authentication error was found.";
char goodie[] = "Valid Token Found, USBSecure Logon Sucessful.";

message.msg_style = PAM_TEXT_INFO;
message.msg = tellthem;

/*
 * authentication requires we know who the user wants to be
 */

retval = pam_get_user(pamh, &user, NULL);
if (retval != PAM_SUCCESS) {
        D(("get user returned error: %s", pam_strerror(pamh,retval)));
        return retval;
}

if (user == NULL || *user == '\0') {
        D(("username not known"));
        retval = pam_set_item(pamh, PAM_USER, (const void *) DEFAULT_USER);
        if (retval != PAM_SUCCESS)
            return PAM_USER_UNKNOWN;
}

//
*****************************************************************************
//check for a deny file, this is /tmp/pam_usbsec.deny
FILE *denyfile = fopen(config.denyfile,"r");
int tokval = 0;

if('k' != (char)fgetc(denyfile)){
        fclose(denyfile);
        tokval = 1;
}

fclose(denyfile);

if(tokval != 0) {
        return PAM_AUTH_ERR;
}
```

```
//
*****************************************************************************
// ok, so now if we are here then usbsec is a valid means and we must check the token

// get the tokenmd5 for the given user

FILE *dbfile = fopen(config.dbfile, "rb");

struct db_struct userdata;
unsigned char tokensum[33];
int tokenfound = 0;

if(!dbfile){
        printf("Error reading db file\n");
        fprintf(logfile, " error reading db file");
        fclose(logfile);
        return 127;
}


while( (fread( &userdata, 1, sizeof( struct db_struct ), dbfile )) > 0  )
{
        fprintf(logfile, "Username: %s\n Token MD5: %s", userdata.username,
        userdata.tokenmd5);


        if(!strcmp(user, userdata.username)){
                strcpy(tokensum, userdata.tokenmd5);
                tokenfound++;
                fprintf(logfile, " found users token\n");
        }

        fprintf(logfile, " and the new tokenmd5 is %s tokenfound %d\n", tokensum,
        tokenfound);


}

user = NULL;       /* clean up */

fclose(dbfile);
fclose(logfile);


if(!tokenfound){
        // the token was not found, exit
        return PAM_AUTH_ERR;
}


// the big check here - see if the signatures match
if(do_md5_file(config.tokenfile, tokensum)){
                return PAM_AUTH_ERR;
}

// if we reach here, the user is authenticated successfully
message.msg = goodie;

if (pam_get_item(pamh, PAM_CONV, (const void **)&conversation) == PAM_SUCCESS &&
conversation) {
       conversation->conv(1, (const struct pam_message **)&pmessage, &resp,
       conversation->appdata_ptr);
```

```
                if (resp) _pam_drop_reply(resp, 1);
        }
        return PAM_SUCCESS;
}

// ***************************************************************************

PAM_EXTERN
int pam_sm_setcred(pam_handle_t *pamh,int flags,int argc
                   ,const char **argv)
{
     return PAM_SUCCESS;
}

/* --- account management functions --- */

PAM_EXTERN
int pam_sm_acct_mgmt(pam_handle_t *pamh,int flags,int argc
                     ,const char **argv)
{
     return PAM_SUCCESS;
}

/* --- password management --- */

PAM_EXTERN
int pam_sm_chauthtok(pam_handle_t *pamh,int flags,int argc
                     ,const char **argv)
{
     return PAM_SUCCESS;
}

/* --- session management --- */

PAM_EXTERN
int pam_sm_open_session(pam_handle_t *pamh,int flags,int argc
                        ,const char **argv)
{
    return PAM_SUCCESS;
}

PAM_EXTERN
int pam_sm_close_session(pam_handle_t *pamh,int flags,int argc
                         ,const char **argv)
{
     return PAM_SUCCESS;
}

/* end of module definition */

#ifdef PAM_STATIC

/* static module data */

struct pam_module _pam_permit_modstruct = {
    "pam_permit",
    pam_sm_authenticate,
    pam_sm_setcred,
    pam_sm_acct_mgmt,
    pam_sm_open_session,
    pam_sm_close_session,
    pam_sm_chauthtok
```

```
};

#endif
```

Listing C.5: md5.c code listing.

```
/*
 *  RFC 1321 compliant MD5 implementation
 *
 *  Copyright (C) 2001-2003  Christophe Devine
 *
 *  This program is free software; you can redistribute it and/or modify
 *  it under the terms of the GNU General Public License as published by
 *  the Free Software Foundation; either version 2 of the License, or
 *  (at your option) any later version.
 *
 *  This program is distributed in the hope that it will be useful,
 *  but WITHOUT ANY WARRANTY; without even the implied warranty of
 *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 *  GNU General Public License for more details.
 *
 *  You should have received a copy of the GNU General Public License
 *  along with this program; if not, write to the Free Software
 *  Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307  USA
 */

#include <string.h>
#include <stdio.h>

#include "md5.h"

#define GET_UINT32(n,b,i)                       \
{                                               \
    (n) = ( (uint32) (b)[(i)    ]       )       \
        | ( (uint32) (b)[(i) + 1] <<  8 )       \
        | ( (uint32) (b)[(i) + 2] << 16 )       \
        | ( (uint32) (b)[(i) + 3] << 24 );      \
}

#define PUT_UINT32(n,b,i)                       \
{                                               \
    (b)[(i)    ] = (uint8) ( (n)       );       \
    (b)[(i) + 1] = (uint8) ( (n) >>  8 );       \
    (b)[(i) + 2] = (uint8) ( (n) >> 16 );       \
    (b)[(i) + 3] = (uint8) ( (n) >> 24 );       \
}

int do_md5_file( char *file, unsigned char *thesum){

        FILE *tokenfile = fopen(file, "rb");

        if(!tokenfile){
                return 127;
        }

        md5_context ctx;
        int i,j;
        char buf[1024];
        unsigned char md5sum[16];

        md5_starts( &ctx );

        while( ( i = fread( buf, 1, sizeof( buf ), tokenfile ) ) > 0 )
```

```
        {
            md5_update( &ctx, buf, i );
        }

        md5_finish( &ctx, md5sum );

        for( j = 0; j < 16; j++ )
        {
                sprintf( buf + j * 2, "%02x", md5sum[j] );
        }

        fclose(tokenfile);

        if( memcmp( buf, thesum, 32 ) )
        {
                return 1;

        } else {
                // md5 was good,
                return 0;
        }


        // if we get here, somethings broked
        return 1;

}

void md5_starts( md5_context *ctx )
{
    ctx->total[0] = 0;
    ctx->total[1] = 0;

    ctx->state[0] = 0x67452301;
    ctx->state[1] = 0xEFCDAB89;
    ctx->state[2] = 0x98BADCFE;
    ctx->state[3] = 0x10325476;
}

void md5_process( md5_context *ctx, uint8 data[64] )
{
    uint32 X[16], A, B, C, D;

    GET_UINT32( X[0],  data,  0 );
    GET_UINT32( X[1],  data,  4 );
    GET_UINT32( X[2],  data,  8 );
    GET_UINT32( X[3],  data, 12 );
    GET_UINT32( X[4],  data, 16 );
    GET_UINT32( X[5],  data, 20 );
    GET_UINT32( X[6],  data, 24 );
    GET_UINT32( X[7],  data, 28 );
    GET_UINT32( X[8],  data, 32 );
    GET_UINT32( X[9],  data, 36 );
    GET_UINT32( X[10], data, 40 );
    GET_UINT32( X[11], data, 44 );
    GET_UINT32( X[12], data, 48 );
    GET_UINT32( X[13], data, 52 );
    GET_UINT32( X[14], data, 56 );
    GET_UINT32( X[15], data, 60 );

#define S(x,n) ((x << n) | ((x & 0xFFFFFFFF) >> (32 - n)))

#define P(a,b,c,d,k,s,t)                                \
```

```
{                                                       \
    a += F(b,c,d) + X[k] + t; a = S(a,s) + b;           \
}

    A = ctx->state[0];
    B = ctx->state[1];
    C = ctx->state[2];
    D = ctx->state[3];

#define F(x,y,z) (z ^ (x & (y ^ z)))

    P( A, B, C, D,  0,  7, 0xD76AA478 );
    P( D, A, B, C,  1, 12, 0xE8C7B756 );
    P( C, D, A, B,  2, 17, 0x242070DB );
    P( B, C, D, A,  3, 22, 0xC1BDCEEE );
    P( A, B, C, D,  4,  7, 0xF57C0FAF );
    P( D, A, B, C,  5, 12, 0x4787C62A );
    P( C, D, A, B,  6, 17, 0xA8304613 );
    P( B, C, D, A,  7, 22, 0xFD469501 );
    P( A, B, C, D,  8,  7, 0x698098D8 );
    P( D, A, B, C,  9, 12, 0x8B44F7AF );
    P( C, D, A, B, 10, 17, 0xFFFF5BB1 );
    P( B, C, D, A, 11, 22, 0x895CD7BE );
    P( A, B, C, D, 12,  7, 0x6B901122 );
    P( D, A, B, C, 13, 12, 0xFD987193 );
    P( C, D, A, B, 14, 17, 0xA679438E );
    P( B, C, D, A, 15, 22, 0x49B40821 );

#undef F

#define F(x,y,z) (y ^ (z & (x ^ y)))

    P( A, B, C, D,  1,  5, 0xF61E2562 );
    P( D, A, B, C,  6,  9, 0xC040B340 );
    P( C, D, A, B, 11, 14, 0x265E5A51 );
    P( B, C, D, A,  0, 20, 0xE9B6C7AA );
    P( A, B, C, D,  5,  5, 0xD62F105D );
    P( D, A, B, C, 10,  9, 0x02441453 );
    P( C, D, A, B, 15, 14, 0xD8A1E681 );
    P( B, C, D, A,  4, 20, 0xE7D3FBC8 );
    P( A, B, C, D,  9,  5, 0x21E1CDE6 );
    P( D, A, B, C, 14,  9, 0xC33707D6 );
    P( C, D, A, B,  3, 14, 0xF4D50D87 );
    P( B, C, D, A,  8, 20, 0x455A14ED );
    P( A, B, C, D, 13,  5, 0xA9E3E905 );
    P( D, A, B, C,  2,  9, 0xFCEFA3F8 );
    P( C, D, A, B,  7, 14, 0x676F02D9 );
    P( B, C, D, A, 12, 20, 0x8D2A4C8A );

#undef F

#define F(x,y,z) (x ^ y ^ z)

    P( A, B, C, D,  5,  4, 0xFFFA3942 );
    P( D, A, B, C,  8, 11, 0x8771F681 );
    P( C, D, A, B, 11, 16, 0x6D9D6122 );
    P( B, C, D, A, 14, 23, 0xFDE5380C );
    P( A, B, C, D,  1,  4, 0xA4BEEA44 );
    P( D, A, B, C,  4, 11, 0x4BDECFA9 );
    P( C, D, A, B,  7, 16, 0xF6BB4B60 );
    P( B, C, D, A, 10, 23, 0xBEBFBC70 );
    P( A, B, C, D, 13,  4, 0x289B7EC6 );
    P( D, A, B, C,  0, 11, 0xEAA127FA );
```

```
    P( C, D, A, B,  3, 16, 0xD4EF3085 );
    P( B, C, D, A,  6, 23, 0x04881D05 );
    P( A, B, C, D,  9,  4, 0xD9D4D039 );
    P( D, A, B, C, 12, 11, 0xE6DB99E5 );
    P( C, D, A, B, 15, 16, 0x1FA27CF8 );
    P( B, C, D, A,  2, 23, 0xC4AC5665 );

#undef F

#define F(x,y,z) (y ^ (x | ~z))

    P( A, B, C, D,  0,  6, 0xF4292244 );
    P( D, A, B, C,  7, 10, 0x432AFF97 );
    P( C, D, A, B, 14, 15, 0xAB9423A7 );
    P( B, C, D, A,  5, 21, 0xFC93A039 );
    P( A, B, C, D, 12,  6, 0x655B59C3 );
    P( D, A, B, C,  3, 10, 0x8F0CCC92 );
    P( C, D, A, B, 10, 15, 0xFFEFF47D );
    P( B, C, D, A,  1, 21, 0x85845DD1 );
    P( A, B, C, D,  8,  6, 0x6FA87E4F );
    P( D, A, B, C, 15, 10, 0xFE2CE6E0 );
    P( C, D, A, B,  6, 15, 0xA3014314 );
    P( B, C, D, A, 13, 21, 0x4E0811A1 );
    P( A, B, C, D,  4,  6, 0xF7537E82 );
    P( D, A, B, C, 11, 10, 0xBD3AF235 );
    P( C, D, A, B,  2, 15, 0x2AD7D2BB );
    P( B, C, D, A,  9, 21, 0xEB86D391 );

#undef F

    ctx->state[0] += A;
    ctx->state[1] += B;
    ctx->state[2] += C;
    ctx->state[3] += D;
}

void md5_update( md5_context *ctx, uint8 *input, uint32 length )
{
    uint32 left, fill;

    if( ! length ) return;

    left = ctx->total[0] & 0x3F;
    fill = 64 - left;

    ctx->total[0] += length;
    ctx->total[0] &= 0xFFFFFFFF;

    if( ctx->total[0] < length )
        ctx->total[1]++;

    if( left && length >= fill )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, fill );
        md5_process( ctx, ctx->buffer );
        length -= fill;
        input  += fill;
        left = 0;
    }

    while( length >= 64 )
    {
```

```
        md5_process( ctx, input );
        length -= 64;
        input  += 64;
    }

    if( length )
    {
        memcpy( (void *) (ctx->buffer + left),
                (void *) input, length );
    }
}

static uint8 md5_padding[64] =
{
 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
    0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
};

void md5_finish( md5_context *ctx, uint8 digest[16] )
{
    uint32 last, padn;
    uint32 high, low;
    uint8 msglen[8];

    high = ( ctx->total[0] >> 29 )
         | ( ctx->total[1] <<  3 );
    low  = ( ctx->total[0] <<  3 );

    PUT_UINT32( low,  msglen, 0 );
    PUT_UINT32( high, msglen, 4 );

    last = ctx->total[0] & 0x3F;
    padn = ( last < 56 ) ? ( 56 - last ) : ( 120 - last );

    md5_update( ctx, md5_padding, padn );
    md5_update( ctx, msglen, 8 );

    PUT_UINT32( ctx->state[0], digest,  0 );
    PUT_UINT32( ctx->state[1], digest,  4 );
    PUT_UINT32( ctx->state[2], digest,  8 );
    PUT_UINT32( ctx->state[3], digest, 12 );
}

#ifdef TEST

#include <stdlib.h>
#include <stdio.h>

/*
 * those are the standard RFC 1321 test vectors
 */

static char *msg[] =
{
    "",
    "a",
    "abc",
    "message_digest",
    "abcdefghijklmnopqrstuvwxyz",
    "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789",
    "12345678901234567890123456789012345678901234567890123456789012" \
```

```
        "345678901234567890"
};

static char *val[] =
{
    "d41d8cd98f00b204e9800998ecf8427e",
    "0cc175b9c0f1b6a831c399e269772661",
    "900150983cd24fb0d6963f7d28e17f72",
    "f96b697d7cb7938d525a2f31aaf161d0",
    "c3fcd3d76192e4007dfb496cca67e13b",
    "d174ab98d277d9f5a5611c2c9f419d9f",
    "57edf4a22be3c955ac49da2e2107b67a"
};

int main( int argc, char *argv[] )
{
    FILE *f;
    int i, j;
    char output[33];
    md5_context ctx;
    unsigned char buf[1000];
    unsigned char md5sum[16];

    if( argc < 2 )
    {
        printf( "\n MD5 Validation Tests:\n\n" );

        for( i = 0; i < 7; i++ )
        {
            printf( " Test %d ", i + 1 );

            md5_starts( &ctx );
            md5_update( &ctx, (uint8 *) msg[i], strlen( msg[i] ) );
            md5_finish( &ctx, md5sum );

            for( j = 0; j < 16; j++ )
            {
                sprintf( output + j * 2, "%02x", md5sum[j] );
            }

            if( memcmp( output, val[i], 32 ) )
            {
                printf( "failed!\n" );
                return( 1 );
            }

            printf( "passed.\n" );
        }

        printf( "\n" );
    }
    else
    {
        if( ! ( f = fopen( argv[1], "rb" ) ) )
        {
            perror( "fopen" );
            return( 1 );
        }

        md5_starts( &ctx );

        while( ( i = fread( buf, 1, sizeof( buf ), f ) ) > 0 )
        {
```

```
            md5_update( &ctx, buf, i );
        }

        md5_finish( &ctx, md5sum );

        for( j = 0; j < 16; j++ )
        {
            printf( "%02x", md5sum[j] );
        }

        printf( "  %s\n", argv[1] );
    }

    return( 0 );
}

#endif
```

Listing C.6: mkconfig.c code listing.

```
// mkconfig - makes a conf file for pam_usbsec

// By Matthew Quarisa - Q11215969

#include <stdio.h>
#include <string.h>
#include "conf.h"

int main(void){

        // make a config structure
        struct usb_config_struct config;

        // now set the paths
        // developer default paths for testing

        strcpy(config.tokenfile, "/tmp/token");
        strcpy(config.denyfile, "/tmp/pam_usbsec.deny");
        strcpy(config.dbfile, "/tmp/usbsec.db");

        // and finally write the structure out as a binary file
        FILE *conffile = fopen(CONFFILE, "wb");

        if( (fwrite( &config, sizeof( struct usb_config_struct ), 1, conffile ) ) != 1){
                printf("Error Writing Structure data!\n");
        } else {
                printf("File written sucessfully.\n");
        }

        fclose(conffile);
        return 0;
}
```

Listing C.7: addtoken.c code listing.

```
// addtoken - adds a token to the database
//
// By Matthew Quarisa - Q11215969

#include <stdio.h>
#include "conf.h"
#include "md5.h"
```

```
#include "db.h"

int main(void){

        struct db_struct userdata;

        //
        ********************************************************************************
        // here we read the config file, it is a binary one with a config struct
        struct usb_config_struct config;
        FILE *conffile = fopen(CONFFILE, "rb");

        if( (fread( &config, 1, sizeof( struct usb_config_struct ), conffile ) ) < 1 )
        {
            // error reading file
            printf("Error loading usbsecure conf file\n");
            return 1;

        }

        printf("Using db path as %s\n", config.dbfile);
        printf("Make sure the USB device has the token created and is connected.\n");

        // ******************************************************************************
        // now open it and md5 the file
        FILE *tokenfile = fopen(config.tokenfile, "rb");

        if(!tokenfile){
                return 127;
        }

        md5_context ctx;
        int i,j;
        char buf[1024];
        unsigned char md5sum[16];

        md5_starts( &ctx );

        while( ( i = fread( buf, 1, sizeof( buf ), tokenfile ) ) > 0 )
        {
            md5_update( &ctx, buf, i );
        }

        md5_finish( &ctx, md5sum );

        for( j = 0; j < 16; j++ )
        {
                sprintf( buf + j * 2, "%02x", md5sum[j] );
        }

        fclose(tokenfile);

        strcpy(userdata.tokenmd5, buf);

        // then ask for the corresponding username
        printf("Please enter the corresponding UNIX username EXACTLY as entered into the
        system: ");
        fgets(userdata.username, 127, stdin);

        if(userdata.username[strlen(userdata.username)-1] == '\n'){
                userdata.username[strlen(userdata.username)-1] = '\0';
        }
```

```
        // then save the two in a struct and write it to the end of the dbfile
        FILE *dbfile = fopen(config.dbfile, "ab");

        if(dbfile){
                if( (fwrite( &userdata, sizeof( struct db_struct ), 1, dbfile ) ) != 1){
                        printf("Error Writing user data!\n");
                } else {
                        printf("Userdata written (hopefully) sucessfully.\n");
                }
        } else {
                printf("Error opening file\n");
        }

        fclose(conffile);
        fclose(dbfile);

        return 0;
}
```

Listing C.8: viewdb.c code listing.

```
// viewdb - dumps the contents of the usbsecure db
//
// By Matthew Quarisa - Q11215969


#include <stdio.h>
#include "conf.h"
#include "md5.h"
#include "db.h"

// note, we should try using args here later
int main(void){

        struct db_struct userdata;

        //
        ********************************************************************************
        // here we read the config file, it is a binary one with a config struct
        struct usb_config_struct config;
        FILE *conffile = fopen(CONFFILE, "rb");

        if( (fread( &config, 1, sizeof( struct usb_config_struct ), conffile ) ) < 1 )
        {
            // error reading file
            printf("Error loading usbsecure conf file\n");
            return 1;

        }

        printf("Using db path as %s\n", config.dbfile);

        // now load the file struct by struct and dump the data
        FILE *dbfile = fopen(config.dbfile, "rb");

        if(!dbfile){
                printf("Error reading db file\n");
                return 127;
        }

        while( fread( &userdata, 1, sizeof( struct db_struct ), dbfile ) )
        {
                printf("Username: %s\n Token MD5: %s", userdata.username, userdata.tokenmd5);
```

```
                if(!do_md5_file(config.tokenfile, userdata.tokenmd5)){
                        printf(" <- Currently Attached.\n");
                } else {
                        printf("\n");
                }

        }

        fclose(dbfile);
        fclose(conffile);

        return 0;
}
```